MODULARITY AND FUNCTIONAL CORRECTNESS EVALUATION OF DESIGN PATTERNS

BY

MAWAL ALI ABDO MOHAMMED

A Thesis Presented to the
DEANSHIP OF GRADUATE STUDIES

KING FAHD UNIVERSITY OF PETROLEUM & MINERALS

DHAHRAN, SAUDI ARABIA

In Partial Fulfillment of the
Requirements for the Degree of

MASTER OF SCIENCE

In

COMPUTER SCIENCE

May, 2014

KING FAHD UNIVERSITY OF PETROLEUM & MINERALS

DHAHRAN- 31261, SAUDI ARABIA

DEANSHIP OF GRADUATE STUDIES

This thesis, written by **Mawal Ali Abdo Mohammed** under the direction his thesis advisor and approved by his thesis committee, has been presented and accepted by the Dean of Graduate Studies, in partial fulfillment of the requirements for the degree of **MASTER OF SCIENCE IN COMPUTER SCIENCE.**

Dr. Mahmoud Elish
(Advisor)

Dr. Adel F. Ahmed
Department Chairman

Dr. Jameleddine Hassine
(Member)

Dr. Salam A. Zummo
Dean of Graduate Studies

Dr. Khaled Al Jasser
(Member)

19/5/14

Date

عمادة الدراسات العليا
DEANSHIP OF GRADUATE STUDIES

DEDICATED TO

I dedicate this thesis with all of my love to Allah

first, then to my parents, my wife, my brother

and sisters.

ACKNOWLEDGMENTS

All praise and glory to Almighty Allah (Subhanahu Wa Taalaa) who gave me courage and patience to carry out this work. Peace and blessing of Allah be upon last Prophet Muhammad (Peace Be upon Him).

With much appreciation and gratitude I thank the following individuals who dedicated themselves to the successful completion of this thesis.

In the first place I would like to record my gratitude to **Dr. Mahmoud Elish** for his supervision, advice, guidance and making the time for my thesis an unforgettable experience. Thanks are due to my thesis committee members **Dr. Jameleddine Hassine** and **Dr. Khaled Al Jasser** for their cooperation, comments and help. Thanks are also due to the Chairman of Information and Computer Science Department **Dr. Adel Ahmed** for providing all the available facilities.

I would like to thank **King Fahd University of Petroleum & Minerals (KFUPM)** for supporting this research. Also, I would like to thank **Taiz University** in Yemen for their support in pursuing master degree.

I owe thanks to my colleagues and my friends for their help and support. A three of them are Mr. Rashad Othman, Bashar Al Hozaim and Mr. Mohanned Albaz and many others; all of whom I will not be able to name here.

Finally, I want to thank my parents and my wife whose prayers are always a great source of strength for me.

Table of Contents

LIST OF TABELS

LIST OF FIGURES

LIST OF ABBREVIATIONS

DPs	:	Design Patterns
CK	:	Chidamber and Kemerer
AUC	:	Area Under the Curve
GoF	:	Gang of Four
CCR	:	Correct Classification Rate
MAE	:	Mean Absolute Error
RMSE	:	Root Mean Squared Error
SDAE	:	Standard Deviation of Absolute Error
CBO	:	Coupling Between Object
LCOM	:	Lack of COhesion Metric

ABSTRACT

Full Name : Mawal Ali Abdo Mohammed

Thesis Title : Modularity and Functional Correctness Evaluation of Design patterns

Major Field : Computer Science

Date of Degree : May 2014

Software quality has been a subject of extensive research in the last two decades. One special area of research is the study of the impact of design patterns on software quality attributes. The impact of design patterns has been evaluated on many quality attributes in the literature. However, not all the quality attributes have been evaluated. In addition to that, there is no consensus among the different studies on the impact of design patterns on quality attributes. The objective of this thesis is to evaluate the modularity (coupling and cohesion) and functional correctness (fault proneness and fault density) of design patterns and to assess the impact of some design patterns metrics on fault prediction. The obtained results show that the classes that participate in the design patterns are more coupled and less cohesive (less modular) than the non-participant classes on all levels (i.e. design level, category level, pattern level and role level). Also, the obtained results show that there is no significant difference in fault-proneness and fault-density when the classes that participate in the design patterns are evaluated against the non-participant classes. However, the classes that participate in the structural design patterns are of less fault proneness and density than the classes that participate in the other categories and the non-participant classes. For the classes that participate in the creational design patterns, it was

found that there is no clear tendency for fault proneness and density compared to the non-participant classes. Also, it was found that there is no significant difference between the classes that participate in the behavioral design patterns and the non-participant classes. The obtained results in evaluating the effectiveness of design patterns metrics with respect to fault prediction show that the AUC (Area Under the Curve) values associated with these metrics are less than 0.7 so they are not practical. It is concluded that the design patterns have a negative impact on modularity. However, this degradation in modularity is necessary for their functions. Also, it is concluded that the use of structural design patterns have a positive impact on functional correctness. Moreover, the evaluated design patterns metrics are found to be useless in fault prediction.

ملخص الرسالة

الاسم الكامل: موال على عبده محمد

عنوان الرسالة: تقييم التبلور والصحة الوظيفية لإنماط التصميم

التخصص: علوم حاسب الآلي

تاريخ الدرجة العلمية: مايو ٢٠١٤

جودة البرمجيات كانت ومازالت موضوع للبحث المتعمق خلال العقدين الماضيين. أحد فروع هذا المجال هو دراسة العلاقة بين خصائص جودة البرمجيات وأنماط التصميم. بالرغم من أن تاثير هذه الانماط قد تم دراسته على بعض خصائص جودة البرمجيات الا ان هذه الدراسات لم تتطرق الى دراسة تأثير جميع هذه الانماط على جميع خصائص جودة البرمجيات. فبعض الانماط لم يتم دراستها وبعض الخصائص لم يتم دراستها ايضا. بالاضافة الى ذلك، فهذه الدراسات توصلت الى نتائج غير متوافقة مع بعضها البعض. من هنا تنبع اهمية المزيد من الدراسات في هذا المجال. الهدف الرئيسي من هذه الدراسة هو تقييم التبلور(الترابط والتماسك) والصحة الوظيفية (العرضة للخطاء و كثافة الأخطاء) لإنماط التصميم. اضافة الى ذلك سنقوم بتقييم كفاءة بعض المقاييس الخاصة بإنماط التصميم في عملية التنباء بالأخطاء. من خلال النتائج التي حصلنا عليها تبين أن الاصناف التي تنتمي الى انماط التصميم اكثر ترابطا واقل تماسكا - مما يعني انها اقل تبلورا - من الاصناف التي لا تنتمي الى أنماط التصميم على جميع المستويات (مستوى التصميم، مستوى الفئة، مستوى النمط ومستوى الدور). و فيما يخص العلاقة بين أنماط التصميم والعرضة للخطاء وكثافة الاخطاء، فلم نجد فرق معتبر على مستوى التصميم. ولكننا وجدنا فرق معتبر عند مقارنة الفئات المختلفة مع بعضها البعض. فقد وجدنا أن فئة الانماط البنيوية أقل عرضة للخطاء و أقل كثافة للأخطاء من الفئات الأخرى ومن الاصناف التي لاتنتمي الى أنماط التصميم. وبالنسبة للاصناف التي تنتمي للفئات الاخرى (فئة الانماط الانشائية وفئة الانماط السلوكية)، فالنتائج التي حصلنا عليها لم تظهر إي فروق معتبرة بينها وبين الأصناف التي لاتنتمي الى أنماط التصميم. أما فيما يخص تقييم كفاءة بعض مقاييس أنماط التصميم في التنباء بالأخطاء فقد وجدنا ان قيمة المساحة تحت المنحي (AUC) المصاحبة لهذه المقاييس اقل من ٠,٧ مما يعني ان هذه المقاييس غير عملية في

التنباء بالاخطاء. و مما توصلنا اليه في هذه الدراسة، يمكننا ان نستنتج ان انماط التصميم ذات تاثير سلبي على التبلور. ولكن يجدر الاشارة الى أن هذا الانحدار في التبلور ضروري لبنية هذه الأنماط حتى تعمل على الوجه المطلوب. ويمكننا ايضا ان نستنتج أن إستخدام الأنماط البنيوية ذو تأثير إيجابي على العرضة للخطاء وكثافة الأخطاء. بالاضافة الى ماسبق نستنتج ايضا أن مقاييس الانماط غير ذات جدوى في التنباء بالأخطاء.

CHAPTER 1

INTRODUCTION

1.1 Introduction

Design Patterns (DPs) are generic solutions to common design problems. The objective of cataloging these solutions is to make them reusable. Gemma et al. classified DPs (known as GoF DPs) into three categories: creational patterns, structural patterns and behavioral patterns [1]. Under each one of these categories, there is a set of design patterns and each design pattern has one or more participating classes.

Since the introduction of DPs, they have attracted the attention of software researchers and practitioners due to the claimed advantages of their application. One advantage is that they are claimed to help improving programmers' productivity. Also, they are claimed to help professional software designers by promoting best practices. Moreover, they are claimed to help novice designers to acquire more experience in software design. Furthermore, they are claimed to help making communication easier among team members. However, these advantages have not been fully evaluated empirically and need to be investigated further. [2]

The impact of DPs on software quality is still a debatable issue as well. The common belief is that the application of DPs results in a higher quality design and consequently a higher quality software [1] [3]. However, this has not been fully investigated empirically.

In addition to that, some other studies suggest that the application of DPs has a negative impact on software quality [4] [5] [6]. For example, there is a common claim that the proper application of DPs in software design reduces the number of faults [7]. This claim has not been fully investigated and some studies suggest that this claim is not necessarily true [8] [9].

Software quality is defined as the possession of the software to certain attributes [10]. These attributes are divided into two categories: internal attributes and external attributes. Modularity is an example for the internal attributes and functional correctness is an example for the external attributes.

Modular design is a design which consists of modules such as classes and packages. These modules interact with each other to achieve the purpose of the system. A good modular design is a design in which the modules of the system are of low coupling and high cohesion. Coupling is the degree to which a module interacts with other modules of the system. Cohesion is the degree to which the pieces of that module are coherent. There are many degrees of cohesion such as procedural and informational cohesion and here are many levels of coupling for instance content and control coupling. For each degree of cohesion and for each level of coupling, a number of measures can be used to measure coupling and cohesion.

The functional correctness, according to the ISO 25010 standard of product software quality, is defined as the "degree to which a product or a system provides the correct results with the needed degree of precision" [11]. In ISO 25010 standard, the software product quality is defined by eight attributes. One of those attributes is the functional suitability. A sub-characteristic of functional suitability is the functional correctness. The

functional correctness can then be measured in terms of fault-proneness and fault density. Class fault-proneness is measured by measuring whether the class is faulty or not and class fault-density is measured by dividing the number of faults in each class by the number of lines of code (LOC) in that class.

In this work, the impact of design patterns on modularity and functional correctness is investigated. In addition to that, the effectiveness of some design patterns metrics is assessed with respect to their performance in fault-prediction.

1.2 Research Problem

Software design is a human artifact and this artifact is a combination of software constructs such as classes and patterns. As a construct in the software design, design patterns may play an important role in the quality of software design.

In the context of software design, producing a good quality design is a major concern for software designers. The study of impact of design patterns on these attributes can provide deeper insights into their impact on the overall quality of the design. However, the impact of design patterns on the different quality attributes has not been fully investigated and need to be investigated further due to the following reasons:

- There is no consensus among the conducted studies on the impact of design patterns on quality attributes.
- Not all the quality attributes are addressed.
- Not all the design patterns are addressed.
- Not all the levels (i.e. design level, category level, pattern level and role level) are evaluated.

1.3 Research Motivation

The evaluation of modularity and functional correctness of design patterns can provide software developers with valuable knowledge. This knowledge can help the software designer in producing better designs. It can provide tips and guidelines for the software designers to help them in the right application of design patterns which, in turns, help in producing better designs. Also, the software testers can utilize this knowledge. It can provide them with a deeper insight to the design. This insight can help the system tester to focus on the troublesome parts of the design that requires more attention. Therefore, system testers can write more useful test cases that address the real issues of the design. The modularity of design patterns also plays an important role in understanding the overall software quality. Modularity plays an important role in software maintenance. It plays an important role in understandability and modifiability of software design. It can provide us with deeper understanding on what to expect in terms of maintainability and in terms of understandability and modifiability.

1.4 Research Objectives

There are three research objectives of conducting this research as follows:

1. To quantitatively assess and compare the modularity of design patterns in object oriented systems. This can be broken down into the following:
    a. Empirically assessing and comparing the coupling and cohesion of participant versus non-participant classes in the design patterns.

b. Empirically assessing and comparing the coupling and cohesion of the classes in each category of the design patterns (creational, structural, and behavioral).

c. Empirically assessing and comparing the coupling and cohesion of the classes in each design pattern.

d. Empirically assessing and comparing the coupling and cohesion of the classes that participate in the different roles of each design pattern.

2. To quantitatively assess and compare the functional correctness of design patterns in object oriented systems. This can be broken down into the following:

    a. Empirically assessing and comparing the fault-proneness and fault density of participant versus non-participant classes in the design patterns.

    b. Empirically assessing and comparing the fault-proneness and fault density of the classes in each category of the design patterns (creational, structural, and behavioral).

    c. Empirically assessing and comparing the fault-proneness and fault density of the classes in each design pattern.

    d. Empirically assessing and comparing the fault-proneness and fault density of the classes that participate in the different roles of each design pattern.

3. Assessing the effectiveness of some design patterns metrics in the performance of faults prediction models.

1.5 Research Questions

To accomplish the objectives of this work, modularity and functional correctness are evaluated in four granularity levels. These levels are:

- **Design level**

In this level, the differences in modularity and functional correctness between the classes that participate in any design pattern and the classes that do not participate in any design pattern are evaluated.

- **Category level**

In this level, the differences in modularity and functional correctness among the classes that participate in the different categories of design patterns are evaluated.

- **Pattern level**

In this level, the differences in modularity and function correctness between the classes that participate in each single pattern with the classes that do not participate in that pattern are evaluated. For example, the difference between the classes that participate in the adapter design pattern and the classes that do not participate in the adapter design pattern is evaluated.

- **Role level**

In this level, the differences in modularity and functional correctness among the different roles of each design pattern are evaluated.

Based on these granularity levels, the following research questions are identified to be answered as a result for this work.

RQ1: Are the classes that participate in design patterns more (coupled, cohesive, fault-prone, fault-dense) than the non-participant classes?

RQ2: is there a significant difference in (coupling, cohesion, fault-proneness, and fault-density) among the classes that participate in the different categories of design patterns?

RQ3: Are the classes that participate in each design pattern more (coupled, cohesive, fault-prone, fault-dense) than the non-participant classes in that pattern?

RQ4: is there a significant difference in (coupling, cohesion, fault-proneness, and fault-density) among the classes that participate in the different roles of each design pattern?

The fifth question is associated with evaluating the effectiveness of design patterns metrics in fault-prediction:

RQ5: Are the design patterns metrics effective in class faults-prediction?

1.6 Research Methodology

In this section, the proposed research approach will be described as follows:

Phase 1: Comprehensive literature review

A comprehensive literature review will be conducted to survey the existence empirical evidence that addressed the quality of design patterns specially class modularity and fault proneness and density.

Phase 2: Identifying and comparing popular design patterns detection tools

The main objective of this phase is to identify a tool or more that help in detecting instances of design patterns. To do so, the available popular tools will be identified. Then these tools will be evaluated. After that one or more of these tools will be chosen to be used in this research.

Phase 3: Data collection and preparation

After conducting a survey for the literature for the existing empirical evidence and tools, the data needed for conducting this study will be collected and prepared. A group of java open source systems will be prepared for this study. For each class in these systems, the faults data will be collected and prepared.

Phase 4: conducting an empirical study to quantitatively assess the modularity (coupling and cohesion) of design patterns in object oriented systems

In this empirical study the modularity of design patterns will be evaluated. First, the classes in the subject systems will be divided into two groups: A group of participant classes and group of non-participant classes in the design patterns. Then for each group, the modularity of classes will be evaluated and compared. After that the modularity of each category of the design patterns (creational, structural and behavioral) will be evaluated and compared. Finally the modularity of each pattern and the modularity of its roles will be evaluated and compared to the other patterns.

Phase 5: Conducting an empirical study to quantitatively assess the functional correctness (fault-proneness and fault density) of design patterns in object oriented systems.

In this empirical study the functional correctness of design patterns will be evaluated. First, for the same two groups of classes in phase 4: the functional correctness of classes will be evaluated and compared. After that the functional correctness of each category of design patterns (creational, structural and behavioral) will be evaluated and compared. Finally the functional correctness of each pattern and the functional correctness of its roles will be evaluated and compared to the other patterns.

Phase 6: Conducting a comparative study to assess the effectiveness of some design patterns metrics on the performance of fault prediction models.

The effectiveness of some design pattern metrics in predicting the fault proneness and the fault density will be evaluated. To do so, prediction models using logistic and linear regressions will be constructed with C & K metrics [12] and design pattern metrics. These models will be assessed and compared.

1.7 Thesis Organization

In the next chapter, the technical background for this thesis is presented. The literature review is presented in chapter 3. In chapter 4, the experimental setup is described. The experimental results are presented in chapter 5 and then these results are summarized and discussed in chapter 6. Finally, the conclusion and the future work is presented in chapter 7.

CHAPTER 2

BACKGROUND

Before proceeding with functional correctness and modularity evaluation, a brief description for the addressed design patterns is presented in this chapter. In depth design patterns descriptions are available in the (GoF) book [1]. After that, a brief description for the available design patterns detection tools with an assessment for their usefulness in this work is provided.

2.1 Design Patterns Description

2.1.1 Creational Design Patterns

Creational design patterns are design patterns that deal with creating objects so that the created objects serve some purpose and they are used to handle certain situation.

2.1.1.1 Builder

It decouples the complex object creation from its representation. This can help in replicating the same creation process for creating different representations. There are 4 roles for the classes that participate in the Builder design pattern as shown in figure 2.1. These roles are: (1) Builder; (2) ConcreteBuilder; (3) Director; (4) Product.

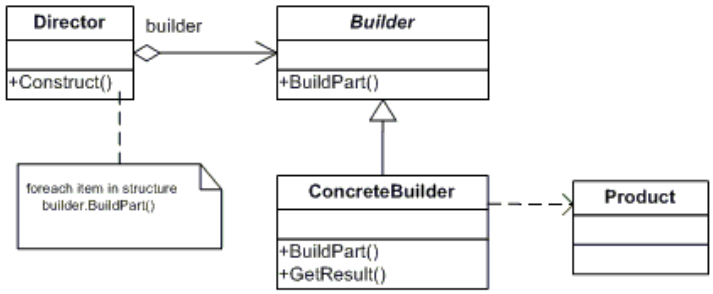


Figure 2.1: Class Structure of Builder Design Pattern

2.1.1.2 Factory Method

The Factory Method design pattern creates an interface for object creation without specifying the exact class of this object. It gives the mission of specifying the class of this object to the subclasses. As it can be seen in figure 2.2, the Factory Method design pattern consists of many classes. These classes play different roles. These roles are as follows: (1) Product; (2) ConcreteProduct; (3) Creator; (4) ConcreteCreator.

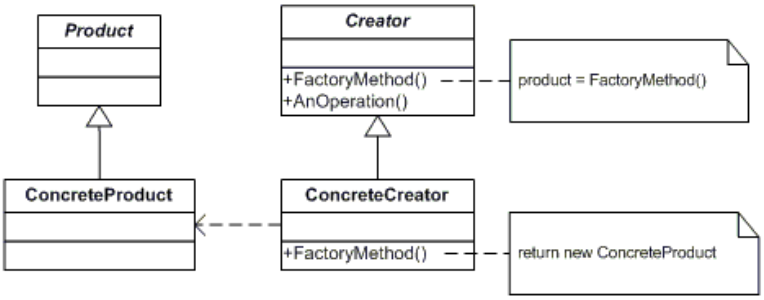


Figure 2.2: Class Structure of Factory Method Design Pattern

2.1.1.3 Prototype

The Prototype design pattern is used when there is a need to create an object that is determined by a prototypical instance. The Prototype pattern composed of many classes

as it can be seen in figure 2.3. These classes play the different roles of the Prototype pattern. These roles are as follows: (1) Prototype; (2) ConcretePrototype; (3) Client.

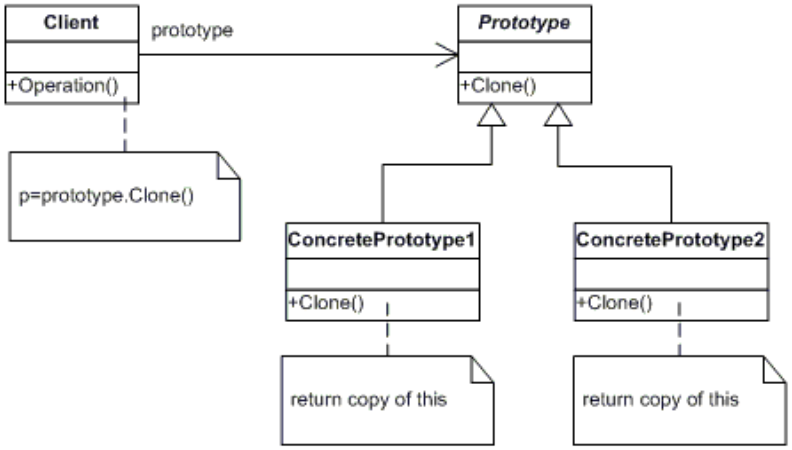


Figure 2.3: Class Structure of Prototype Design Pattern

2.1.1.4 Singleton

Singleton design pattern is a one class design pattern as it can be seen in figure 2.4. The objective of using this pattern is to ensure that there is only one instance of a class and providing a global point to access it.

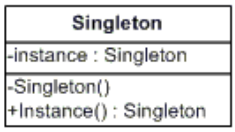


Figure 2.4: Class Structure of Singleton Design Pattern

2.1.2 Structural Design Patterns

Structural design patterns are design techniques that facilitate software design by identifying simple ways to realize relationships among the different entities.

2.1.2.1 Adapter

The Adapter design pattern lets the incompatible interfaces works together. It converts the interface of a class so that it can be used by the client. The Adapter design pattern consists of many classes as it can be seen in figure 2.5. Each class plays a different role in the pattern. There are four roles that the Adapter classes are playing. These roles are as follows: (1) Target; (2) Client; (3) Adaptee; (4) Adapter.

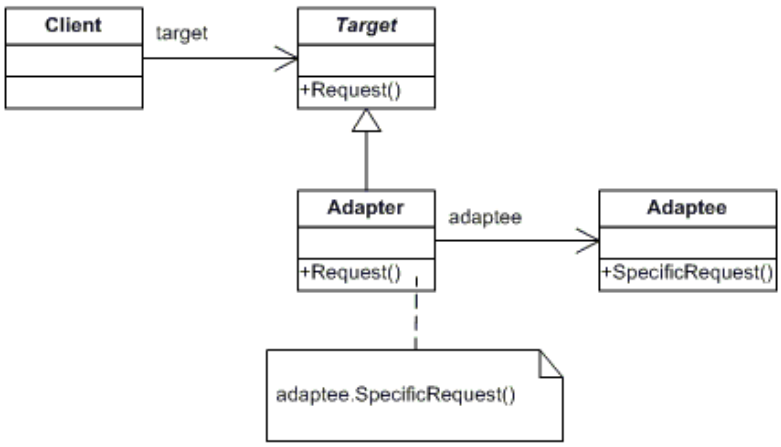


Figure 2.5: Class Structure of Adapter Design Pattern

2.1.2.2 Bridge

The Bridge design pattern separates implementation form abstraction so that each one of them can be manipulated independently. It has many classes as it can be seen in figure 2.6. Each class plays a different role. There are four different roles. These roles are as follows: (1) Abstraction; (2) RefinedAbstraction; (3) Implementor; (4) ConcreteImplementor.

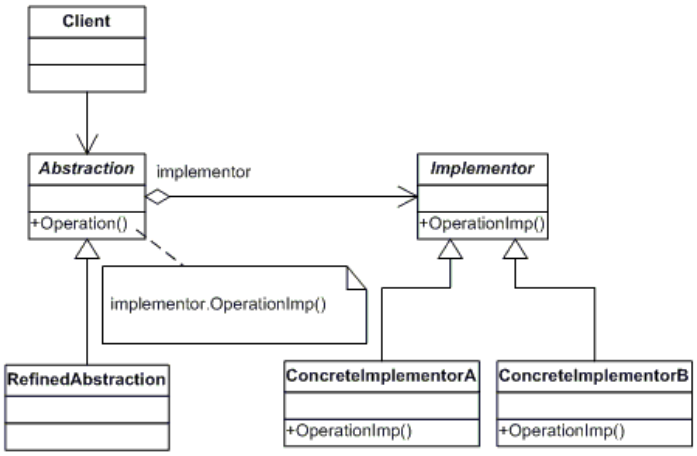


Figure 2.6: Class Structure of Bridge Design Pattern

2.1.2.3 Composite

The Composite design pattern composes a part-whole hierarchy of objects. This hierarchy helps treat these objects in the same way as if they were a single instance so that the client treats all objects uniformly. As it can be seen in figure 2.7, the Composite design pattern consists of many classes. These classes play different roles in this pattern. These roles are: (1) Component; (2) Leaf; (3) Composite; (4) Client.

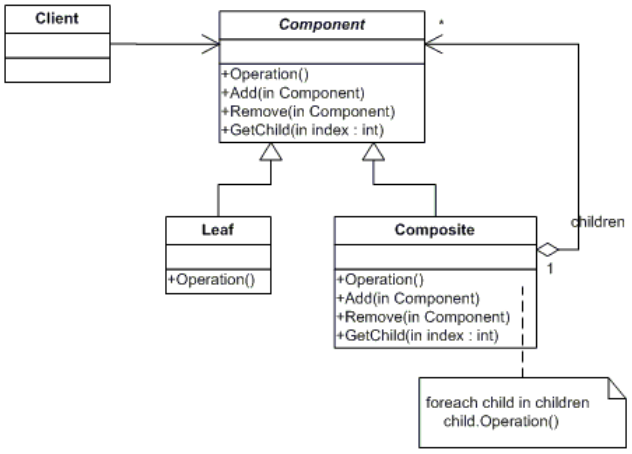


Figure 2.7: Class Structure of Composite Design Pattern

2.1.2.4 Decorator

The Decorator design pattern helps extend functionality of an object dynamically. It does that through attaching additional responsibilities to that object and providing a flexible alternative to sub-classing. The Decorator design pattern consists of many classes as it can be seen in figure 2.8. These classes play different roles. These roles are as follows: (1) Component; (2) ConcreteComponent; (3) Decorator; (4) ConcreteDecorator.

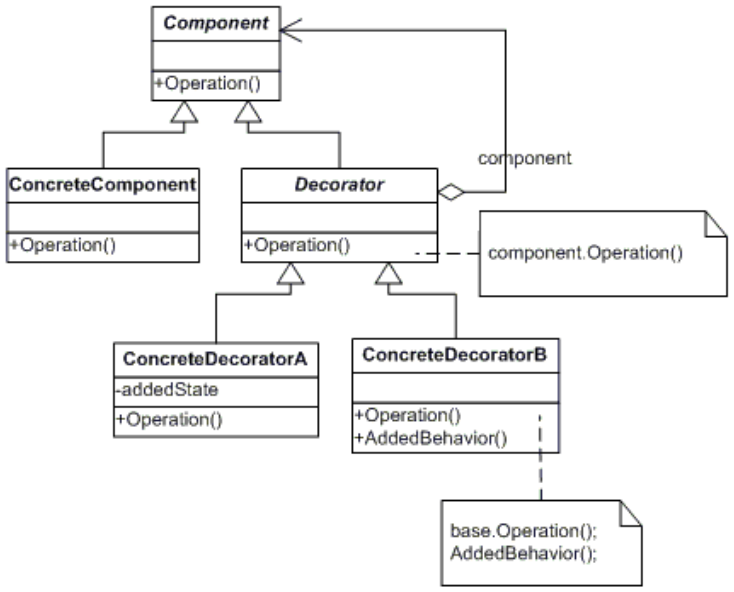


Figure 2.8: Class Structure of Decorator Design Pattern

2.1.2.5 Proxy

The Proxy design pattern work as a surrogate to another object. The objective of that is to control access to that object. There are many classes participate in the Proxy design pattern as it can be seen in figure 2.9. These classes play 3 roles in the Proxy design pattern. These roles are as follows: (1) Proxy; (2) Subject; (3) RealSubject.

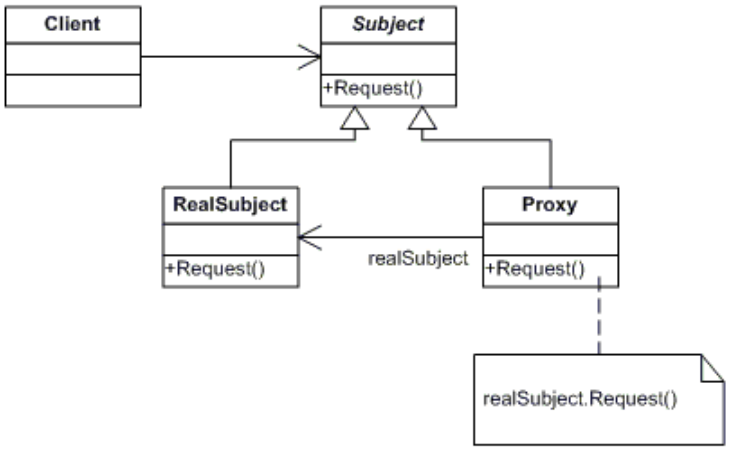


Figure 2.9: Class Structure of Proxy Design Pattern

2.1.3 Behavioral Design Patterns

Behavioral design patterns are communication patterns. These patterns ease and increase the flexibility of communication.

2.1.3.1 Command

In the Command design pattern, an object is used to encapsulate a request to a method. These requests can be used as parameters to different clients. As it can be seen in figure 2.10, the Command design pattern consists of many classes. These classes play different roles. These roles are as follows: (1) Command; (2) Concrete-Command; (3) Client; (4) Invoker; (5) Receiver.

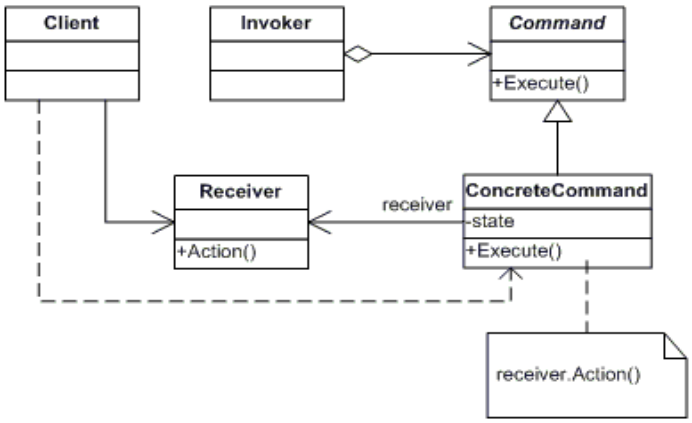


Figure 2.10: Class Structure of Command Design Pattern

2.1.3.2 Iterator

The Iterator design pattern provides a way to traverse the aggregate object's elements sequentially. It decouples the aggregate object from the algorithms. As it can be seen in figure 2.11, the Iterator design pattern consists of many classes. These classes play different roles. These roles are as follows: (1) Iterator; (2) Concrete-Iterator; (3) Aggregate; (4) Concrete-Aggregate.

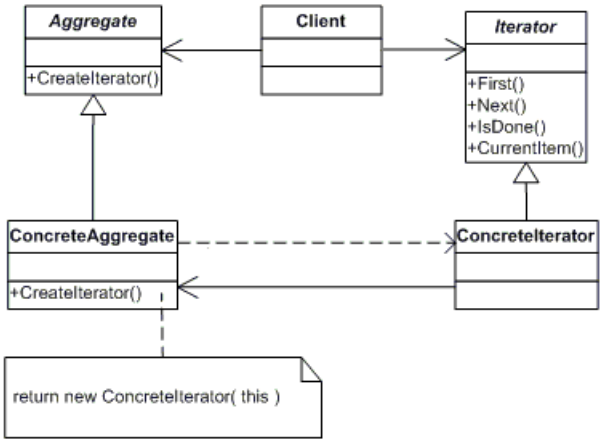


Figure 2.11: Class Structure of Iterator Design Pattern

2.1.3.3 Memento

The Memento design pattern is used to encapsulate the internal state of an object so than it can be restored later. The Memento design pattern consists of many classes as it can be seen in figure 2.12. These classes participating in the Memento design pattern play different roles. These roles are: (1) Originator; (2) Memento; (3) Caretaker.

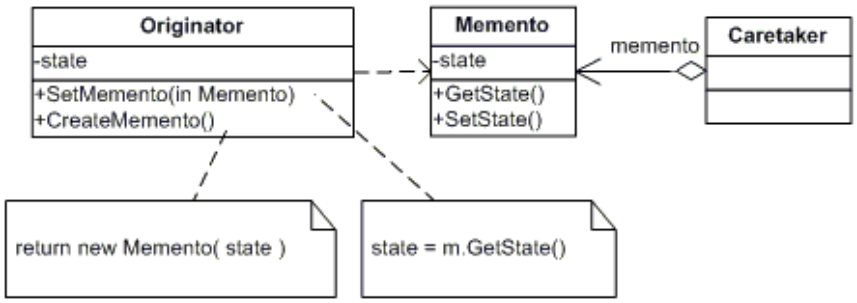


Figure 2.12: Class Structure of Memento Design Pattern

2.1.3.4 Observer

The Observer design pattern maintains 1: M dependency among objects. This dependency utilized by notifying the dependent objects of any change in the state of the independent object. As it can be seen in figure 2.13 that there are many classes participate in the Observer design pattern. These classes participate in the different roles of Observer design pattern. These roles are follows: (1) Subject; (2) Concrete-Subject; (3) Observer; (4) Concrete-Observer.

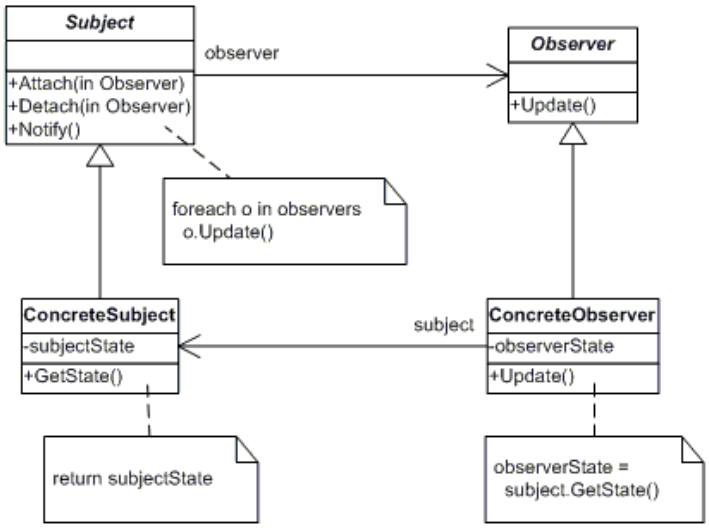


Figure 2.13: Class Structure of Observer Design Pattern

2.1.3.5 State

The State design pattern allows for different behaviors based on the change in the object internal state. The State design pattern consists of many classes as it can be seen in figure 2.14. These classes play different roles in the State design pattern. These roles are: (1) Context; (2) State; (3) Concrete-State.

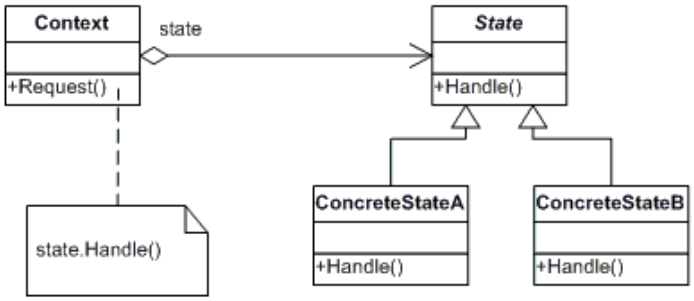


Figure 2.14: Class Structure of State Design Pattern

2.1.3.6 Strategy

In the Strategy design pattern a set of algorithms are defined and encapsulated in different classes. This encapsulation allows of different clients to use the different algorithms independently. As it can be seen in figure 2.15, the Strategy design pattern consists of many classes. These classes play 3 roles. These roles are as follows: (1) Strategy; (2) Concrete-Strategy; (3) Context.

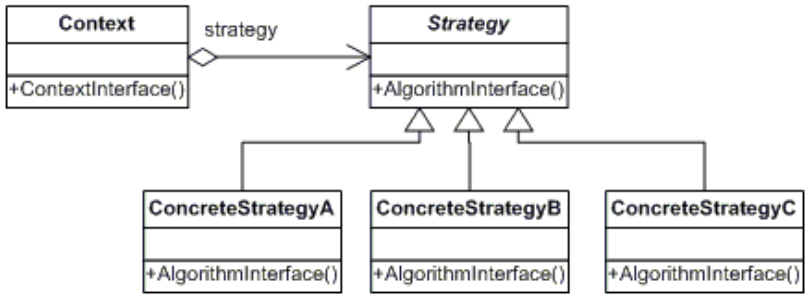


Figure 2.15: Class Structure of Strategy Design Pattern

2.1.3.7 Template Method

Template Method design pattern defines a general structure of an algorithm in a class but without determining specific steps in this algorithm. These steps are determined later in the subclasses. This allows defining different variations of the algorithm without changing the structure of the algorithm. The Template Method design pattern consists of two classes as it can be seen in figure 2.16. These classes play two roles: (1) Abstract-Class; (2) Concrete-Class.

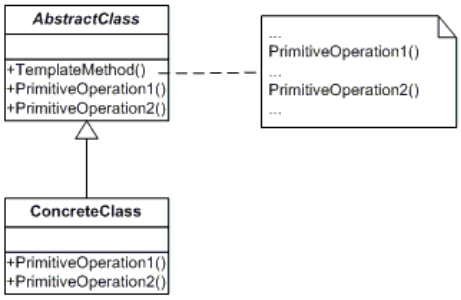


Figure 2.16: Class Structure of Template Method Design Pattern

2.1.3.8 Visitor

If there is a set of elements of an object structure on which the operations of a class can be performed and we want to add a new operation to it, the Visitor design pattern facilitates adding new operation to this class. The newly added operation in the Visitor design pattern will be performed on the same set of elements of the object structure without changing the class. As it can be seen in figure 2.17, the Visitor design pattern consists of many classes. These classes play different roles. These roles are as follows: (1) Visitor; (2) Concrete-Visitor; (3) Element; (4) Concrete-Element; (5) Object-Structure.

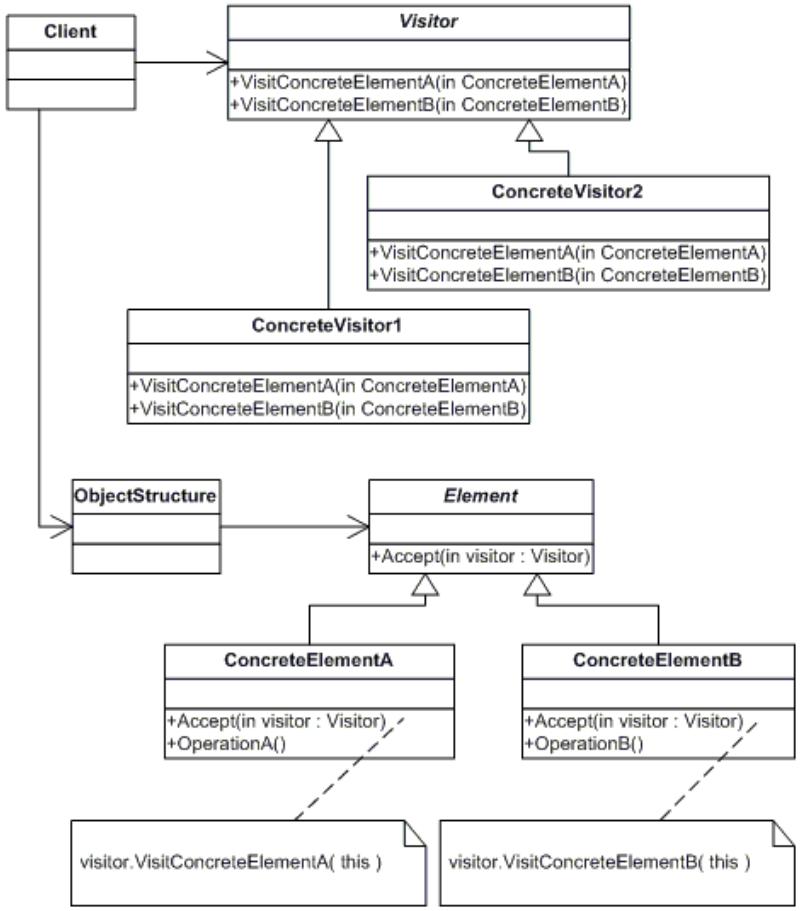


Figure 2.17: Class Structure of Visitor Design Pattern

2.2 Identification and Comparison of Design Patterns Detection Tools

There are many characteristics that should be available in the tool or in the combination of tools that is/are going to be used to collect pattern data information for our work. These characteristics are as follows:

1- Be able to work on Java source code since that the subject systems are in java and their faults data are available.

2- At least half of the patterns in each category should be covered. This is because that one of our objectives of conducting this study is to compare the difference in modularity and functional correctness among the different categories of software design patterns.
3- Be able to collect the roles information of the different design patterns. This is because one objective of this work is to assess the difference in modularity and functional correctness among the different roles in each design patterns.
4- A high level of detection accuracy is required (at least 95%). This is because a low level of detection accuracy will affect the planned experiments negatively.

The literature is surveyed searching for software design patterns detection tools. As a result, 10 tools that work on java language source code are identified. These tools are as follows:

2.2.1 DeMIMA

It is a semi-automatic tool for identification of pattern-like micro-architecture in java source code. This tool ensures the traceability of the pattern like micro-architecture between design and implementation. There are three layers involved in the process of DeMIMA. The first and the second layer are to recover an abstract model from the source code. The third layer is to detect patterns from the recovered abstract model. [13]

In evaluating the suitability of this tool for our work, it was found that this tool is not suitable. It can be seen in table 2.1 that the precision of this tool is too low. This is expected to affect the results of our work. So, this tool will not be considered.

2.2.2 DP-Miner

In this tool, design patterns discovery starts with defining the structural characteristics of each design pattern. These characteristics are represented in terms of weights and matrix. The discovery process includes three major processes: structural analysis, behavioral analysis and semantic analysis. Out of these analyses, the design patterns are retrieved. [14]

In evaluating the suitability of this tool for our work, it was found that this tool is not suitable. As it can be seen in table 2.1, this tool is designed to detect a very few number of design patterns: Adapter/Command, Bridge, Composite and State/Strategy. At the same time, this tool does not differentiate between the instances of Adapter and Command and between the instances of State and Strategy. So, this tool will not be considered in our work.

2.2.3 DPRE

There are two phases in the recovery process in this tool. In the first phase, a coarse grained level recovery process is applied to the source code. In this level, the structure of the design patterns is considered and a parsing technique for visual language is utilized as well. In the second phase, a fine grain validation is applied to the retrieved instances in the first phase. [15]

In evaluating the suitability of this tool for our work, it was found that this tool is not suitable. There are two problems in this tool. As it can be seen in table 2.1, the first one is that this tool detects structural design patterns only. This violates the second characteristic mentioned above. The second problem is that the precision rate of this tool varies from

62% to 97% and the recall rate is not mentioned, as it can be seen in table 2.1. The achieved precision rate is not enough to obtain accurate results and the recall rate is not mentioned to make sure that this tool retrieves only true pattern instances.

2.2.4 FUJABA

This tool utilizes fuzzy logic and abstract syntax graph to recover design patterns. The Abstract syntax graph utilizes to cope with the design variants of design patterns. The different design variants models are modeled with different graphs. The fuzzy logic utilizes to cope with the implementation variants. A set of fuzzy rules are defined together to handle the implementation variants by giving a degree of belief to test whether a pattern is found or not. [16]

In evaluating the suitability of this tool for our work, it was found that this tool is not suitable. The major reason for that is the absence of the tool evaluation. This tool is not evaluated with any performance measures as it can be seen in table 2.1. So, this tool is not suitable for this work.

2.2.5 MARRPLE

Design patterns detection module of this tool consists of four different components. The first component is the information detector engine which collects the required information for pattern detection. The second component is the jointer which extracts the potential design patterns. The third component is the classifier which validates the retrieved design patterns. The last component is the output generator which is responsible for providing the user with the final output. [17]

In evaluating the suitability of this tool for our work, it was found that this tool is not suitable. The reason is that this tool can detect three patterns only: Abstract Factory, Composite and Visitor. This is not suitable for this work as mentioned in the second characteristic of the required tool above. Also, it can be seen in table 2.1 that the performance of this tool is not satisfying.

2.2.6 Pinot

This tool reclassified design patterns into different categories claiming that this reclassification facilitates design pattern detection. They use light-weight static analysis for analyzing the programs. [18]

In evaluating the suitability of this tool for our work, it was found that this tool is not suitable. The reason for that is the absence of precision and recall rates or any other performance measure of this tool as it can be seen in table 2.1. So, this tool is not suitable for this work.

2.2.7 PTIDEJ

This tool uses constraints solving with explanation to detect design patterns. The tool starts with detecting design patterns that exactly match the predefined instances. Then the constraints are relaxed to allow for detecting more patterns. [19]

In evaluating this suitability of this tool for our work, it was found that this tool is not suitable. As it can be seen in table 2.1, the recall and precision rates are absent. So, this tool is not suitable for this work.

2.2.8 Tsantalis' tool

This tool uses the similarity scoring between graph vertices to detect design patterns. It also exploits the fact that a design pattern resides in one or more hierarchy. [20]

In evaluating the suitability of this tool for our work, it was found that this tool is not suitable. There are two reasons for that. The first is that, although this tool can detect 13 patterns, we can only benefit from 9 patterns as it can be seen in table 2.1. This is because this tool cannot differentiate between the instances of Adapter and Command patterns and between the instances of State and Strategy patterns. The second reason is that this tool does not extract all patterns information – this tool cannot extract all the roles in each design pattern.

2.2.9 SPQR

The developers of this tool use a formal denotational semantics to encode design pattern elements and to encode the rules by which these elements are combined to form design patterns. Also, these semantics are used to encode the structural and behavioral relationships among the elements of design patterns. [21]

In evaluating the suitability of this tool for our work, it was found that this tool is not suitable. First of all, this tool can only detect one pattern which is the Decorator as it can be seen in table 2.1. Secondly, this tool does not provide any verification for its performance.

2.2.10 Web of Pattern (WOP)

In this tool an ontology language is used to define design patterns and to create a web of patterns. Then an algorithm for finding exact matches is applied to extract the instances of

design patterns. The constraints can also be relaxed to retrieve more potential instances. [22]

In evaluating the suitability of this tool for our work, it was found that this tool is not suitable. The first reason is that this tool only detects 4 patterns: Abstract Factory, Bridge, Strategy and Adapter. The second reason is that the precision and recall rates are not satisfying as it can be seen in table 2.1.

Table 2.1: Design patterns detection tools Comparison

Tools	Patterns	Precision	Recall
DeMIMA	Abstract Factory, Composite, Adapter, Command, Decorator, Observer, State/Strategy, Prototype, Visitor, Singleton, Template Method and Factory Method	34	100
DP-Miner	Adapter/Command, Bridge, State/Strategy and Composite	91 - 100	97
DPRE	Adapter, Bridge, Composite, Façade, Proxy and Decorator	62 - 97	-
FUJABA	All (GoF) patterns	-	-
MARRPLE	Abstract Factory, Composite and Visitor	78.6	78.3
Pinot	All (GoF) patterns except (Builder, Prototype, Command, Interpreter, Iterator and Memento)	-	-
PTIDEJ	All (GoF) patterns except (Builder, Bridge, Interpreter, Iterator)	-	-
Tsantalis' tool	Singleton, Composite, Adapter/Command, Decorator, Observer, State/Strategy, Prototype, Visitor, Template Method and Factory Method	100	95.9
SPQR	Decorator	-	-
WOP	Abstract Factory, Bridge, Strategy and Adapter.	57.3	54.5

CHAPTER 3

LITERATURE REVIEW

This chapter summarizes the previous related studies in the literature that have evaluated the impact of design patterns on the different software quality attributes.

3.1 Overview of the collected studies

The literature is surveyed looking for the relevant studies to our goal. A set of 16 primary studies were identified. In this section, the primary studies are reviewed and evaluated.

3.1.1 Distribution of Studies by Quality Attributes

Table 3.1 shows the distribution of the primary studies based on the software quality attributes that were explored by them. Four quality attributes have been identified. The majority of studies have been conducted to study the impact of design patterns on software maintainability.

Table 3.1: Distribution of studies by s/w quality attributes

S/W quality attribute	Studies
Maintainability	[23], [24], [25], [26], [27], [28], [29] , [30]
Evolution and change proneness	[31], [32], [33]
Performance	[34], [35]
Faults	[9], [36], [8]

3.1.2 Distribution of Studies by Granularity Level

The primary studies can be also divided into two groups. One group of studies investigated the impact of design patterns on the overall quality of the design (design-

level), whereas the other group investigated the impact on the class-level. The class-level studies investigated the quality attributes for the participant classes versus the non-participant classes in a design pattern. Table 3.2 shows the distribution of these studies accordingly.

Table 3.2: Distribution of studies by level

Level	Studies
Design level	[23], [24], [25], [26], [27], [28], [29] , [30], [31], [34], [35], [9], [36]
Class level	[8], [32], [33]

3.1.3 Quality Attributes Proxy Definitions

Each primary study defines the addressed quality attribute in some way. The proxy definition is used for quantifying the quality attributes to be measurable. Table 3.3 shows the proxy used in each study.

Table 3.3: Quality attributes proxy definitions

Quality attribute	Study #	Proxy
Maintainability	[23], [24], [25], [27], [28]	1- Time required for understanding and modification. 2- Correctness.
	[26]	Time required for understanding and modification.
	[29]	1- Understandability in terms of time, the # of correct answer to some questions and efficiency. 2- Modifiability in terms of time, score of modification and efficiency.
	[30]	The duration of the eye-fixation are used to measure the attention spent to perform comprehension and modification tasks.

Change-proneness	[31], [32], [33]	# of changes
Performance	[35]	# of requests and the processing time of each request.
	[34]	Memory need and Execution time
Faults	[9], [36], [8]	# of faults

3.2 Design Pattern and Quality Attributes

This section summarizes the state of the art studies that have evaluated the impact of design patterns on software quality attributes.

3.2.1 Design Patterns and Maintainability

Prechelt et al. [23] conducted an experiment to study the effect of design patterns on software maintenance using four different subject systems. They addressed five patterns: Decorator, Composite, Abstract Factory, Observer and Visitor. They found that it is preferable to use a design pattern even if there is a simple solution but not in all cases. For the exceptions, where the simple solution is preferred, it is good to follow the software engineer common sense about whether to use a pattern or not, and in case of doubt, it is better to use a pattern as a default approach.

Vokac et al. [24] conducted a replication of the experiment done by Prechelt et al. [23]. They used the same set of subject systems. To increase the realism of their experiment, they conducted their experiment in a real programming environment rather than using paper and pen. The results indicate that it is not good to generalize the effects of software design patterns on maintainability. They showed that each pattern has its own impact on maintainability. For the Observer and Decorator, they found that they can be easily understood with a little or no pattern knowledge. For the Composite, because of its

reliance on recursion, it showed some problems. For the Visitor pattern, it took a lot of time for understanding and the required modifications came with poor correctness.

Other replication experiments for the experiment done by Prechelt et al. [23] were conducted. Prechelt and Liesenberg [25] replicated that experiment, and used two systems out of the four used in the original experiment. Due to the small size of the experiment they found only one statistically significant result: the non-pattern based version of only one system was more maintainable and can be extended easily. Juristo and Vegas [26] also conducted another replication study for Prechelt et al. [23]. They addressed three different patterns: Abstract Factory, Composite and Decorator. They conducted their study on two software systems. Their results contradict with the original study. They found that systems with design patterns were less maintainable. Nanthaamornphong and Carver [27] also conducted a replication study for the same experiment of Prechelt et al. [23]. In their experiments they used the same subject systems of the original experiment. They focused on four patterns (Observer, Visitor, Decorator and Composite); one in each software system. They found that design patterns have no impact on the understandability and maintainability of these systems. Krein et al. [28] performed also a replication for the same experiment done by Prechelt et al. [23]. In this experiment they studied three different patterns: Decorator, Composite and Abstract Factory. They found that by performing some modifications on the two versions, the pattern version and the non-pattern version, the pattern based designs took longer time and have more faults than non-pattern designs except for one modification task.

Garzas et al. [29] investigated the relationship between the design patterns and the maintainability in terms of understandability and modifiability. They addressed three

patterns: State, Composite and Chain of Responsibility. They found that the use of design patterns make the diagrams difficult to understand and require more effort to modify.

Jeanmart et al. [30] studied the impact of Visitor design pattern on both maintainability and comprehensibility. They performed an experiment that measured the maintainability and the comprehensibility in terms of eye-movement and eye-fixation. They tracked the eye-movement using a software system called EyeLink-II. They were assigned some tasks to work on. During performing these tasks, the data in terms of eye-movement and eye-fixation were collected. The study concluded that the Visitor pattern requires more time for comprehension and modification. However, the representation of Visitor pattern in its canonical form reduced the effort for modification tasks.

3.2.2 Design Patterns and Evolution/Change-proneness

Aversano et. al. [31] reported an empirical study on the evolution of software design patterns. In this study, they analyzed the way the pattern changes, what kinds of changes the patterns are subject to and which classes co-change with the change of patterns. The identified patterns in these systems were Observer, Composite Adapter, Command, Decorator, Factory, and Visitor. They found that participant patterns in the implementation of the major requirements of those systems were more subject to change than the other patterns.

Bieman et al. [33] studied the change proneness in the participant versus the non-participant classes in the design patterns. They conducted their study on five different systems. After identifying the design patterns used in each one of the subject systems, they analyzed the changes in many different versions of each system. They found that

classes that participate in software design patterns are more change-prone than non-participant classes.

Gatrell et al. [32] conducted a study on a commercial C# software system, in which, they studied the relationship between the change proneness and the software design patterns. They found that classes that participate in patterns have more tendencies for change than classes that do not participate in design patterns. They found that the Adaptor, Method, Proxy, Singleton, State, Strategy and Visitor patterns associated with most of the changes whereas Command and Creator patterns associated with lower number of changes.

3.2.3 Design Patterns and Performance

Rudzki [35] studied the difference between the impacts of two different design patterns on the performance of a software system. He chose the Command and the Façade design patterns in his study. The reason for choosing these two patterns is that they have similar functionality so they can be used alternatively. By running the system in 9 different test cases, he found that the Façade pattern performed better that the Command pattern.

Afacan [34] studied the impact of State design pattern on memory usage and execution time of popular fixed-point DSP processor manufactured by Texas Instruments. The Results suggested that more memory usage and more execution time are needed by the object oriented system with State design pattern. But for complex systems it is better to use the object oriented with State design pattern which leads to a cleaner architecture, allows reuse of software components and may get gain in comparison with C code duplications in the complex systems.

3.2.4 Design Patterns and Faults

Vokac [36] studied the defect rates and design patterns usage in a large industrial software written in C++. He chose only five patterns for investigation: Singleton, Template Method, Decorator, Observer and Abstract Factory. He found that the Observer and the Singleton design patterns are associated with the larger code structures, and thus require more attention. In addition, the Factory pattern is more compact and loosely coupled, and thus associated with fewer defects. There was no clear tendency for the Template Method pattern as it was used in both simple and complex structures.

Gatrell and Counsell [8] studied the relationship between fault-proneness and design patterns in a subset of a commercial software system, written in C#. They studied 13 design patterns. They found that participant classes were more fault-prone than non-participant classes in the design patterns. Also, they found that the most fault-prone patterns were the Adaptor, Method and Singleton patterns. They justified this by the propensity of participant classes in the design pattern for change.

Ampatzoglou et al. [9] conducted a study to understand the relationships between design pattern application on computer games and defect frequency. They addressed 11 different design patterns. They found that there is no correlation between the number of instances of design patterns and software bugs. However, specific design patterns showed significant impact on software bugs. The adapter design pattern showed negative impact on defect frequency and the observer pattern showed positive impact and it leads to decrease in the number of software defects.

3.3 Evaluation of the collected studies

Table 3.4 provides an overview of the covered design patterns in each study and compares them with the covered design patterns in the other studies. Also, it summarizes the impact of design patterns on the different quality attributes. The plus, minus and equal signs mean that the impact of a design pattern on the corresponding quality attribute is positive (i.e. more maintainable, higher performance, less changes or less faults), negative (i.e. less maintainable, lower performance, more changes or more faults) and neutral respectively. The ± sign means that the associated design pattern has a positive impact in some contexts and a negative impact in other contexts. For "*", it means that the corresponding study provide a general conclusion on the impact of design patterns on the quality attribute. The absence of the "*" means that the corresponding study provides a specific conclusion for each one of the addressed patterns.

Table 3.4: Summary of design patterns coverage and impact on the quality attributes

Pattern type	Quality attribute	Maintainability								Evolution and change-proneness			Performance		Faults		
	Pattern	[23]	[24]	[25]	[26] *	[27] *	[28] *	[29] *	[30] *	[31] *	[32]	[33] *	[34]	[35]	[9]	[36]	[8] *
Creational patterns	Abs. Factory	=	=	-	-		-			-					=		
	Builder										=	-					-
	Factory Method										=	-				+	-
	Prototype														=		
	Singleton										-	-			=	-	-
Structural patterns	Adapter									-	-	-			-		-
	Bridge																
	Composite	=	-	-	-	=	-	-		-					=		
	Decorator	+	+	=	-	=	-			-					=	=	
	Facade													+			
	Flyweight																
	Proxy										-	-			=		-
Behavioral Patterns	Chain of Resp.							-									
	Command									-	+	-		-			-
	Interpreter																
	Iterator										-	-					-
	Mediator																
	Memento																
	Observer	-	+			=				-					+	-	

	State							-			-	-	±		=		-
	Strategy										-	-			=		-
	Template														=	±	
	Visitor	=	-			=			-	-	-	-					-

Many things can be observed in table 3.4:

- Regarding maintainability, a group of 5 patterns was covered well in the literature. These patterns are: Abstract Factory, Composite, Decorator, Observer and Visitor. But we should notice that these patterns were covered by one experiment and its five replications. Also, it can be observed that there is no consensus among these studies on the impact of these patterns on maintainability.
- Also, a group of 16 patterns was never investigated with respect to their impact on software maintainability. Those patterns are: Builder, Factory Method, Prototype, Singleton, Adapter, Bridge, Façade, Flyweight, Proxy, Command, Interpreter, Iterator, Mediator, Memento, Strategy and Template.
- The impact of State and Chain of Responsibility design patterns on maintainability was covered only by one study.
- For the change-proneness and faults, all patterns were covered in the literature except five patterns. The uncovered patterns are: bridge, Façade, Flyweight, interpreter, Mediator and Memento. However, not all of these patterns addressed by each study. Each study covers a different set of patterns.
- For performance, only three patterns were covered in the literature by only two studies. The addressed patterns are: Façade, Command and State.
- We can observe that most of the studies investigated the impact of design patterns generally; they did not differentiate among the different patterns.

3.4 Comparison with the previous works

In this study we evaluate two additional attributes (Modularity and fault-density) which are not addressed in the literature. Also, we provide additional empirical evidence on the impact of design patterns on fault-proneness since that there were no consensus in the literature on the impact of design patterns on fault-proneness. Moreover, we addressed 17 patterns while the maximum number of the addressed patterns in the literature is 11. Furthermore, we evaluate modularity and functional correctness on all level: design level, category level, pattern level and role level. Finally, we empirically assessed whether class design pattern metrics are effective in class fault-prediction or not.

CHAPTER 4

EXPERIMENTAL SETUP

4.1 Data Description

The goal of this work is to evaluate the difference in modularity and functional correctness for the different categories of design patterns and for the different patterns. Also we aim at assess the impact of class participation in design patterns on the presence of faults. The planned experiments will be conducted on five open source software systems. Accordingly, the required data sets that must be available in these systems are as follows:

- Pattern data: which classes participate in which design patterns.
- Number of faults in each class.
- Coupling and cohesion of each class.

4.1.1 Subject Systems Description

In this section, a brief description for the chosen subject systems is provided.

- **JHotDraw v5.1** is a framework for the creation of drawing editors. The creation of geometric and user defined shapes, editing those shapes, creating behavioural constraints in the editor and animation are supported by this framework.
- **JUnit v3.7** is a simple framework for creating test cases that are used repeatedly. It used to write test cases for Java programs.

- **Lexi v0.1.1 alpha** is a word processor. It can be used in editing many files such as RTF and HTML files in addition to the plain text.
- **Nutch v0.4** is an extensible and scalable web crawler. It can be used in searching, indexing and scoring of filers.
- **PMD v1.8** is a source code analyzer. It scans the source code searching for standard coding rules violations and other problems such as suboptimal code and Dead code.

4.1.2 Pattern and Fault Data

The reason for choosing the already described systems is the availability of pattern data in the P-Mart repository [10]. As we explained in chapter 2, the pattern identification and detection tools are not suitable for our work since that all of them are not congruent with the characteristics required to obtain reliable data. So, we decided to search for another source of pattern data. As a result of our search we come across P-Mart repository which is a reliable source of pattern data that has been used in several works [37], [38], [39], [40], [41]. The data in the P-Mart repository was collected based on GoF's book [1]. It consists of 11 systems. Each system patterns collected in a separate session by B.Sc. and M.Sc. students. Also, the collections of other people who are working in this area are used as well.

Table 4.1 reports the number of patterns' instances in each one of the subject systems and table 4.2 provides descriptive statistics for each subject system. We can see in these tables that the number of DPs and the percentages of the participating classes in DPs are different from one system to the other. The number of patterns ranges from 5 to 21 in the

different systems and the percentage of the participating classes ranges from 10.6% to 74%. The same thing can be said about the number and the percentages of faulty classes in the subject systems. The number and the percentages of faulty classes ranges from 9 (11.5%) to 233 (52.2%) in the subject classes. This is even clearer in figure 4.1 and figure 4.2.

We collected faults data from the development log files associated with each subject system. As mentioned before, the subject systems of this study are open-source systems. They are hosted in sourceforge website. For extracting the log file of the JUnit, the following command is used:

"cvs -d:pserver:anonymous@junit.cvs.sourceforge.net:/cvsroot/junit rlog junit".

This command is for extracting the log file of JUnit. For the other systems, the name of the systems is replaced with Junit in this command. After extracting the file, the faults data of each class is collected into an excel file.

Table 4.1: Patterns coverage in the subject systems

Category	Pattern	JHotDraw	JUnit	Lexi	Nutch	PMD
Creational patterns	Abs. Factory					
	Builder			1		2
	Factory Method	3				3
	Prototype	2				
	Singleton	2	2	2	1	
Structural patterns	Adapter	1			2	1
	Bridge				2	
	Composite	1	1			2
	Decorator	1	1			
	Facade					
	Flyweight					

	Proxy					1
Behavioral patterns	Chain of Resp.					
	Command	1			2	
	Interpreter					
	Iterator		1		1	1
	Mediator					
	Memento				2	
	Observer	2	3	2		2
	State	2				
	Strategy	4			2	
	Template	2			3	1
	Visitor					1
Total # of instances	**21**	**8**	**5**	**15**	**14**	

Table 4.2: Descriptive statistics of the subject systems

Systems	**# of classes**	**Total LOC**	**# of faulty classes**	**# of participating classes in DPs**
JHotDraw	155	8891	45 (29%)	115 (74%)
JUnit	78	4773	9 (11.5%)	45 (57.6%)
Lexi	24	7045	11 (44%)	7 (28%)
Nutch	165	23507	74 (44.8%)	22 (13.3%)
PMD	446	41486	233 (52.2%)	49 (10.6%)
All Systems	868	85702	372(42.8%)	238(27.4%)

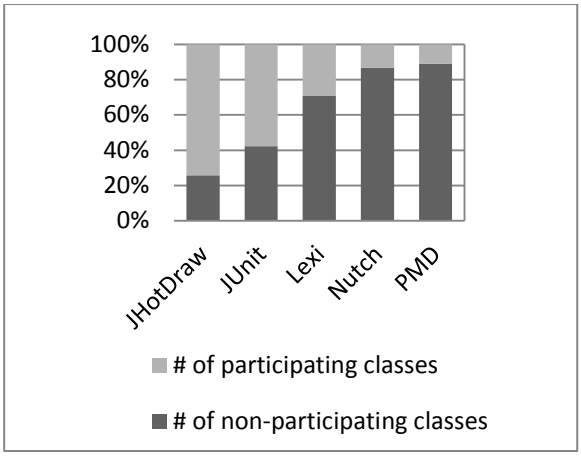


Figure 4.1: Percentages of participant to non-participant classes

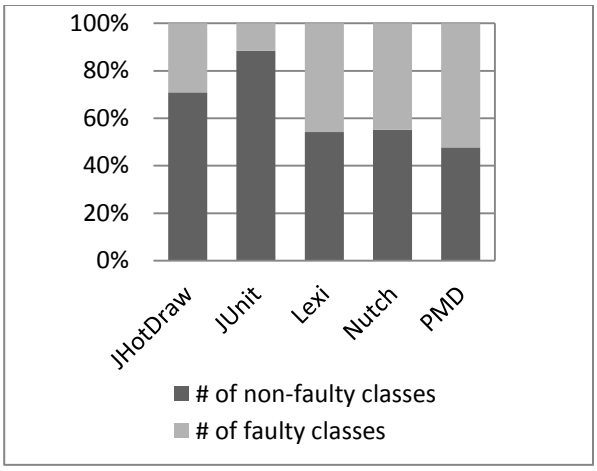


Figure 4.2: Percentages of non-faulty to faulty classes

4.1.3 Coupling and Cohesion Data

There are many metrics for measuring coupling and cohesion of classes. In our research we adopted – CBO and LCOM. The first reason for adopting this metrics is that these metrics is the most widely used. The second is the availability of tools for calculating

these metrics. The tool used in our research to calculate these metrics is Understand from SciTools [42].

- **CBO**

  The Coupling Between Object Classes (CBO) measure for a class is a count of the number of other classes to which it is coupled [12]. Class X is coupled to Class Y if X uses a data type or a member function of the class Y.

- **LCOM**

  The Lack of Cohesion Metric (LCOM) is an object-oriented metric used to measure the cohesiveness of a class [12]. This metric calculate the lack of cohesion. To calculate the cohesion of the class, which what we need, we calculate LCOM and then subtract it from 100%.

4.2 Methodology Description

In order to achieve the objective we planned, we have to decide on how to measure the differences between groups. Also, we need to decide on the prediction models. For measuring the differences between groups, we chose to use Mann-Whitney U test to measure the difference between two groups and the Kruskal test to measure the difference among more than two groups. For prediction models, we use linear regression and logistic regression. We adopt linear regression for fault-density prediction and logistic regression for fault-proneness prediction.

The independent variables in this work are the class participation in the design patterns and the dependent variables are the class (fault-proneness, fault-density, coupling and cohesion).

4.2.1 Mann-Whitney Test

The Mann-Whitney U test is a non-parametric test used to compare differences between two independent groups [43]. This test is used to evaluate the differences between the different groups. For example, it is used to compare the difference in coupling, cohesion, fault-pronenss and fault-density between the class that participate in the design patterns and the non-participant classes. Another example, this test is used to evaluate the difference in coupling, cohesion, fault-density and fault-proneness between the classes that participate in the structural design patterns and the classes that participate in the creational design patterns. Moreover, this test is to evaluate the differences between each pair of roles in each design pattern. There are four assumptions that must be held to perform this test. These assumptions are as follows:

1- Dependent variables should be either ordinal or continuous. In our case the values of fault-proneness are categorical (either 0 or 1) which is treated by this test as ordinal. For coupling and cohesion, their values are ordinal. Regarding fault-density, its value is continuous.

2- Independent variable should be consisting of two categorical groups. In our case the values of the class participation in design pattern is either 0 or 1: 0 refers to the category of non-participating classes and 1 refers to the category of participating classes.

3- There should be an independence of observations. In our case, there is no relationship between the different observations so one observation does not affect another observation (i.e. if a class is a participant in one pattern does not imply or require to be or not to be a participant in another pattern).

4- The two independent variables (i.e. groups) should not be normally distributed. To test the normality of the data, we performed two tests. These tests are Kolmogorov-Smirnov and Shapiro-Wilk. These tests are performed on the coupling, cohesion, fault-density and fault-proneness data of the participant versus the non-participant classes when all systems combined together. The results associated with these tests are in table 4.3. The assumption in evaluating the normality is "the data are not normally distributed" and it was found that the p-value associated with evaluating the normality is less than 0.05. Accordingly we accept the hypothesis.

Table 4.3: Tests of Normality

Data set	Groups	Kolmogorov-Smirnov (Significance)	Shapiro-Wilk (Significance)
LCOM	Non-participant	0.000	0.000
	Participant	0.000	0.000
CBO	Non-participant	0.000	0.000
	Participant	0.000	0.000
FaultDensity	Non-participant	0.000	0.000
	Participant	0.000	0.000
FaultProneness	Non-participant	0.000	0.000
	Participant	0.000	0.000

4.2.2 Kruskal-Wallis Test

Kruskal-Wallis test is a non-parametric test used to compare differences between two or more independent groups [44]. This test is used to evaluate the differences among the different roles in each design pattern. There are two assumptions that must be held to perform this test. These assumptions are as follows:

1- Dependent variables should be either ordinal or continuous. In our case the values of fault-proneness are categorical (either 0 or 1) which is treated by this test as ordinal. For coupling and cohesion, their values are ordinal. Regarding fault-density, its value is continuous.

2- Independent variable should be consisting of more than two categorical groups. In our case this variable consists of more than two groups in all of the addressed patterns except for one which is singleton which consists of one role only. All the other patterns consist of more than two roles.

4.2.3 Logistic regression

Logistic regression is a type of probabilistic statistical classification model. It is used in our work to predict binary response (a class is faulty or not) from a binary predictor (participating or not in a design pattern or in a category). We use logistic regression model to relate an independent variable (e.g. participating or not in a design pattern) to a dependent variable (a class is faulty or not) and to measure the significance of this relationship.

4.2.4 Linear regression

Linear regression attempts to model the relationship between two variables, the dependent and the independent variables, by fitting a linear equation to the observed data. We are going to use linear regression to relate the participation in design patterns information to the fault-density. Participating information in design patterns is the independent variable and the fault- density information is the dependent variable.

CHAPTER 5

EXPERIMENTAL RESULTS

The results of evaluating modularity and functional correctness are presented in this chapter. In addition to that the assessment of the effectiveness of some design patterns metrics to predict faults is presented. Before proceeding with evaluating of modularity and functional correctness, the descriptive statistics for the metrics used in evaluating modularity (coupling and cohesion) and functional correctness (fault-proneness and fault-density) are presented in table 5.1. It is clear that the minimum value in all metrics in all cases is 0. Also, we can see that the maximum value for fault-proneness is 1 in all cases. For the other statistics, they are different from one case to the other.

Table 5.1: Descriptive statistics for CBO, LCOM, Fault-proneness and fault-density

Mertic	Mean	Minimum	Maximum	Std. Dev.
JHotDraw				
CBO	3.89032	0.00	27.0000	4.42919
LCOM	41.20000	0.00	100.0000	38.95792
Fault-proneness	0.29032	0.00	1.0000	0.45538
Fault-density	13.47620	0.00	285.7143	36.12245
JUnit				
CBO	2.28205	0.00	19.0000	3.40701
LCOM	26.30769	0.00	95.0000	33.95616
Fault-proneness	0.11538	0.00	1.0000	0.32155
Fault-density	6.61585	0.00	375.0000	43.01357
Lexi				
CBO	3.04167	0.00	28.00000	7.32761
LCOM	58.20833	0.00	99.00000	36.03619

Fault-proneness	0.45833	0.00	1.00000	0.50898
Fault-density	5.05220	0.00	31.25000	8.38321
Nutch				
CBO	3.88485	0.00	33.0000	5.57696
LCOM	47.33939	0.00	100.0000	34.60835
Fault-proneness	0.44848	0.00	1.0000	0.49885
Fault-density	10.10898	0.00	125.0000	18.91519
PMD				
CBO	4.57399	0.00	97.0000	8.21988
LCOM	23.78924	0.00	100.0000	30.85807
Fault-proneness	0.52466	0.00	1.0000	0.49995
Fault-density	34.14107	0.00	800.0000	62.54685
All systems				
CBO	4.07258	0.00	97.0000	6.85291
LCOM	32.55300	0.00	100.0000	35.18454
Fault-proneness	0.42972	0.00	1.0000	0.49532
Fault-density	22.60484	0.00	800.0000	51.16812

5.1 Modularity and functional correctness evaluation

The evaluation of modularity and functional correctness is conducted in four levels of design patterns as follows:

5.1.1 Design Level

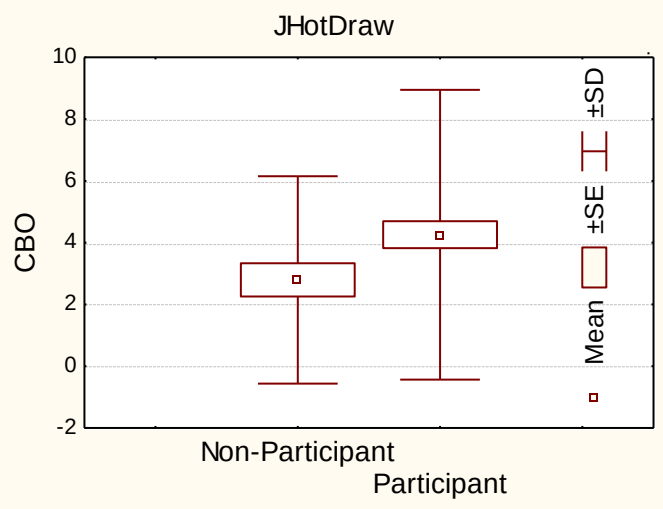
In this level, we evaluate the difference in modularity (Coupling and Cohesion) and functional correctness (fault-density and fault-proneness) between the classes that participate in any design pattern and the classes that do not participate in any design pattern (i.e. Participant vs. Non-participant). This evaluation gives a general insight on the impact of design patterns on modularity and functional correctness.

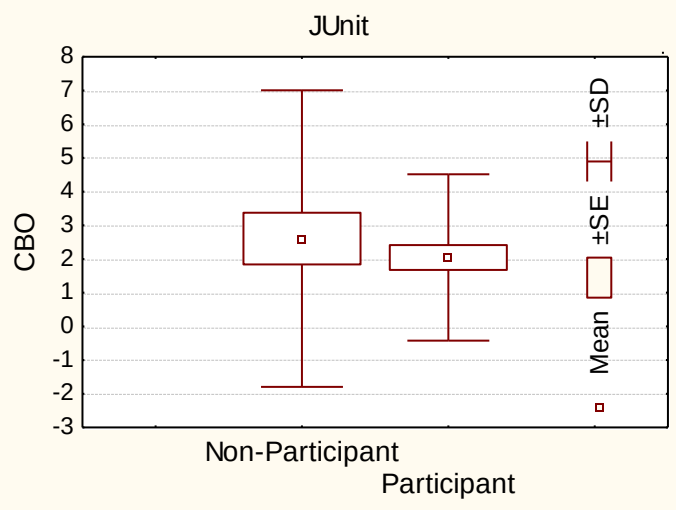
- **Coupling**

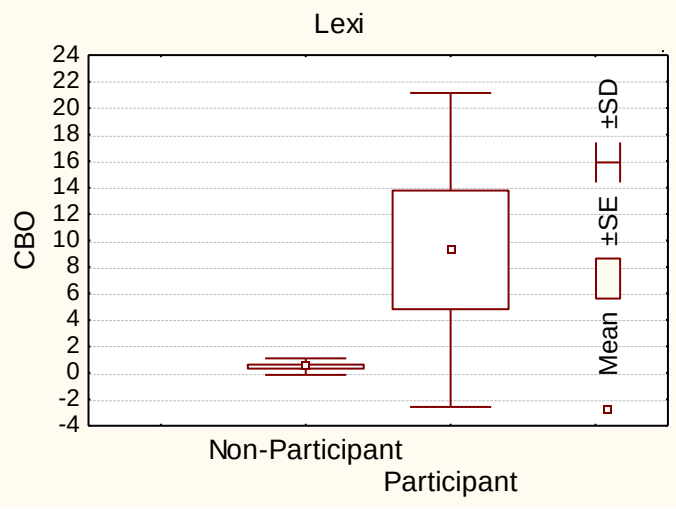
The results of coupling evaluation of the participant versus the non-participant classes are shown in table 5.2. The obtained results show significant differences in each one of the subject systems except for the JUnit. Also, the obtained results show significant difference when we combine all systems together. To investigate the nature of this difference, we need to take a look at figure 5.1. It is clear that in all of the cases, where the p-values are significant, the participant classes are more coupled than the non-participant classes. For JUnit, we can see that it is the other way around (i.e. the participant classes are less coupled than the non-participant classes).

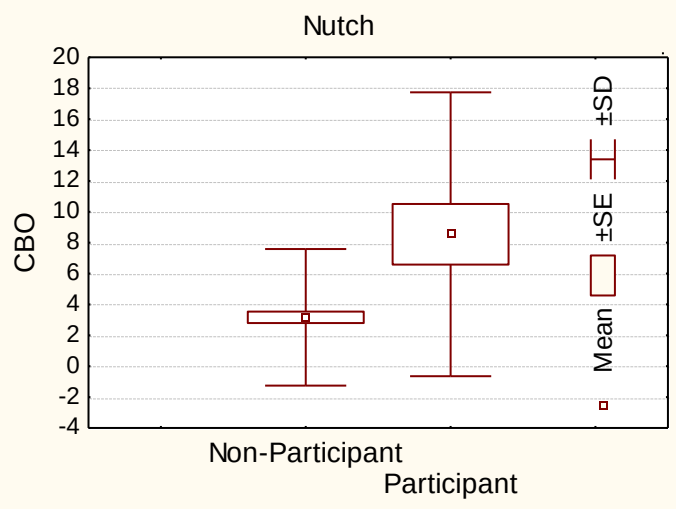
Table 5.2: P-values of Mann-Whitney U test analysis for coupling evaluation of Participant vs. non-participant classes

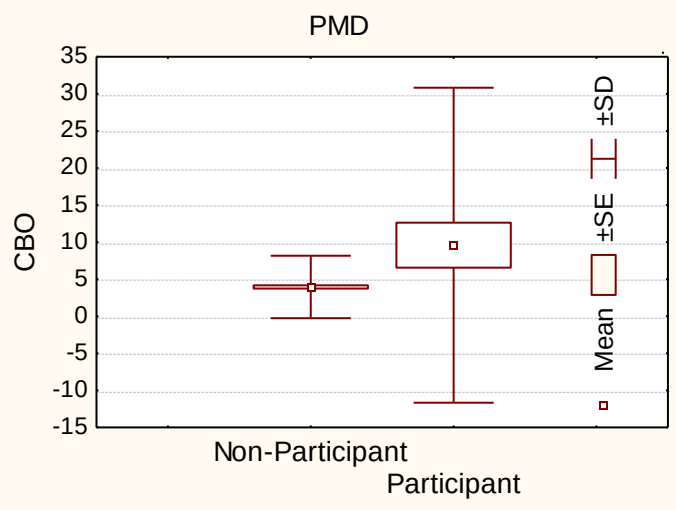
	Participant vs. non-participant
JHotDraw	**0.005314**
JUnit	0.908789
Lexi	**0.000299**
Nutch	**0.005497**
PDM	**0.040875**
All systems	**0.022466**


JHotDraw
CBO
10
8
6
4
2
0
-2
Non-Participant
Participant
±SD
±SE
Mean


JUnit
CBO
8
7
6
5
4
3
2
1
0
-1
-2
-3
Non-Participant
Participant
±SD
±SE
Mean


Lexi
CBO
24
22
20
18
16
14
12
10
8
6
4
2
0
-2
-4
Non-Participant
Participant
±SD
±SE
Mean


Nutch
CBO
20
18
16
14
12
10
8
6
4
2
0
-2
-4
Non-Participant
Participant
±SD
±SE
Mean


PMD
CBO
35
30
25
20
15
10
5
0
-5
-10
-15
Non-Participant
Participant
±SD
±SE
Mean

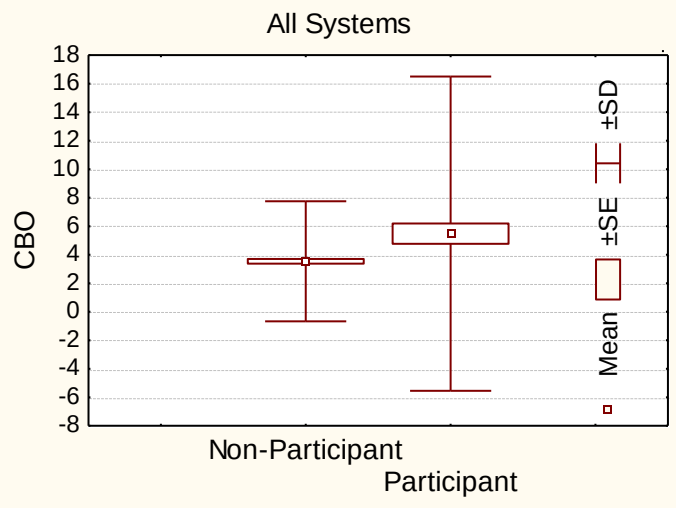

All Systems
CBO
18
16
14
12
10
8
6
4
2
0
-2
-4
-6
-8
Non-Participant
Participant
±SD
±SE
Mean

Figure 5.1: Coupling values comparison for participant versus non-participant classes

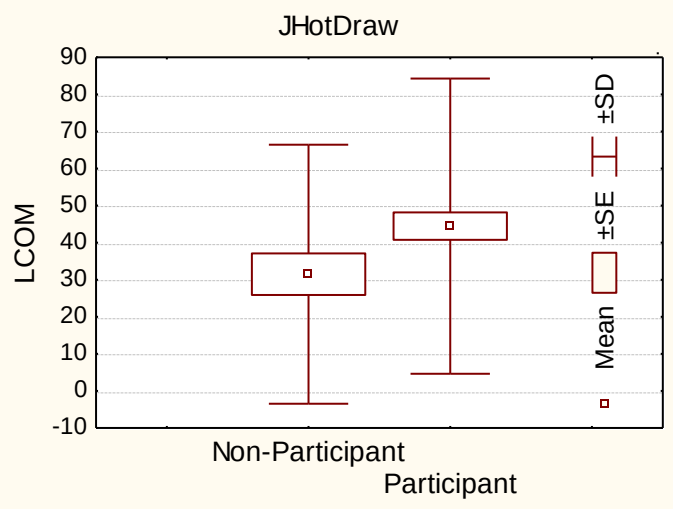
- **Cohesion**

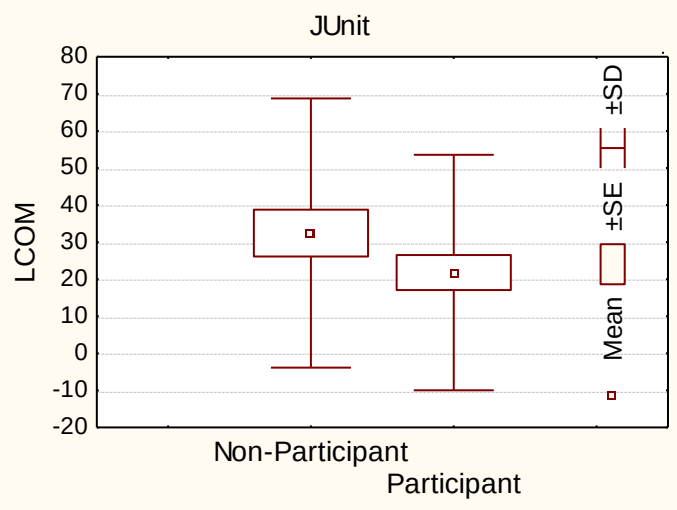
Two p-values are reported as significant in evaluating the cohesion of participant versus the non-participant classes as in table 5.3. These values are obtained in case of Lexi and when all systems combined together. Also, we can see that the p-value associated with JHotDraw is significant in the 90% confidence interval.

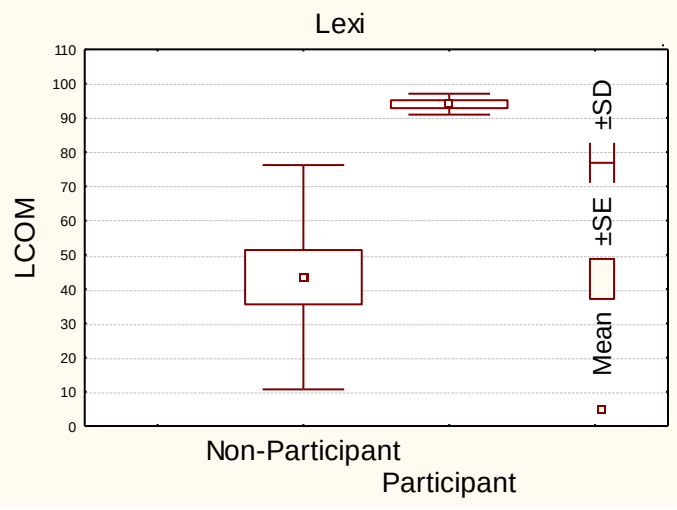
Regarding the tendency of the relationship, as it can be seen in figure 5.2 that the non-participating classes tends to be more cohesive than the participant classes in all cases except for JUnit.

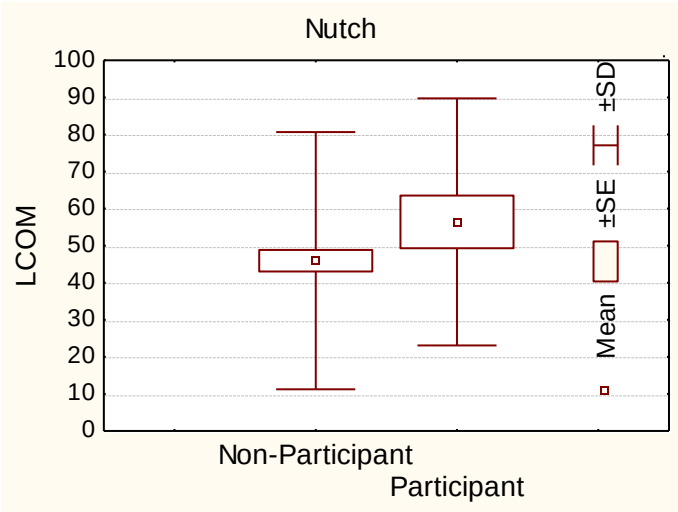
Table 5.3: P-values of Mann-Whitney U test analysis for Cohesion evaluation of Participant vs. non-participant classes

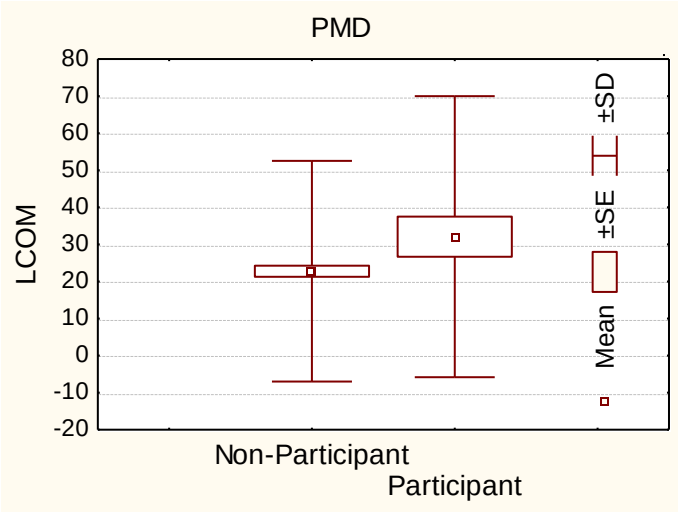
	Participant vs. Non-participant
JHotDraw	0.065895
JUnit	0.246362
Lexi	**0.000484**
Nutch	0.139579
PDM	0.119617
All systems	**0.000185**


JHotDraw
LCOM
90
80
70
60
50
40
30
20
10
0
-10
±SD
±SE
Mean
Non-Participant
Participant


JUnit
LCOM
80
70
60
50
40
30
20
10
0
-10
-20
±SD
±SE
Mean
Non-Participant
Participant


Lexi
LCOM
110
100
90
80
70
60
50
40
30
20
10
0
±SD
±SE
Mean
Non-Participant
Participant





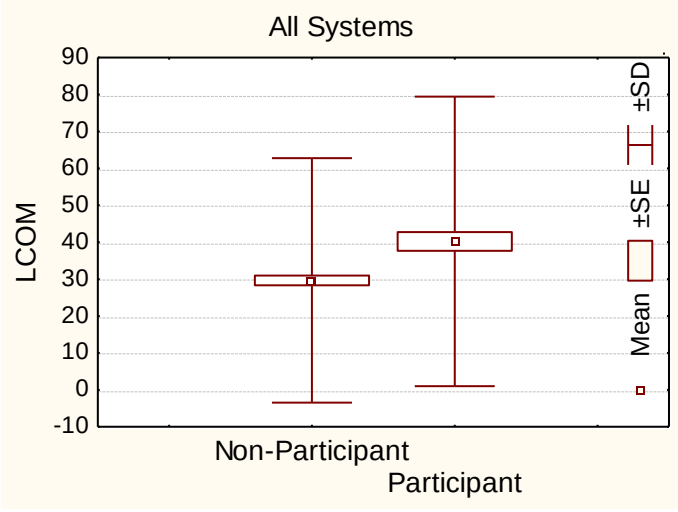


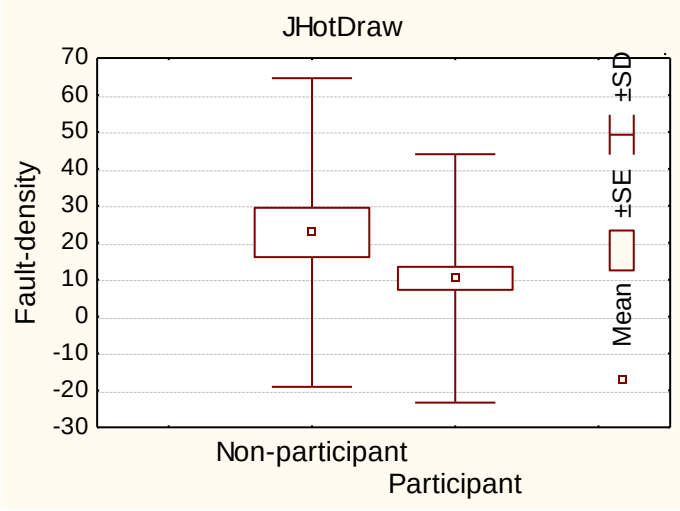
Figure 5.2: Cohesion comparison of participant vs. non-participant classes

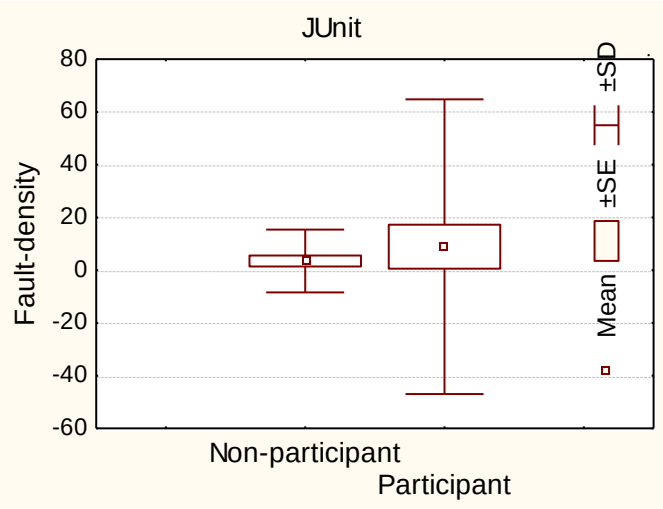
- **Fault-density**

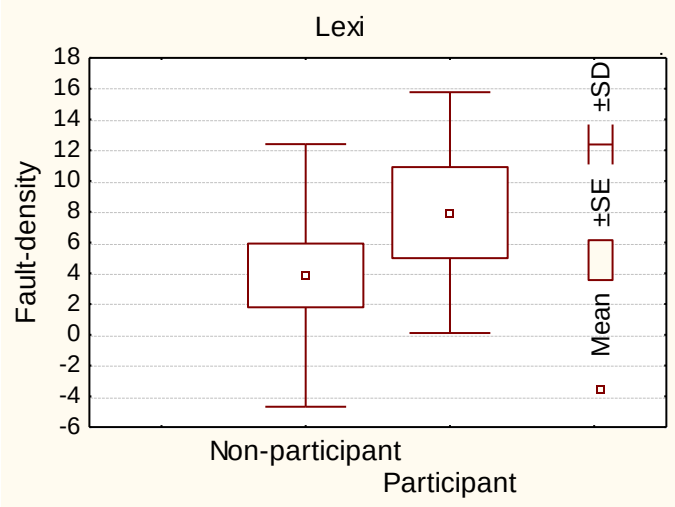
Only two values are reported as significant in evaluating fault-density of participant versus non-participant classes as shown in table 5.4. But, as it can be seen in figure 5.3, the obtained results are in two different directions. The participating classes group in Lexi is more fault-dense than the non-participating classes group but they are less dense than the non-participating classes when all systems combined. We can see from figure 5.3 that the tendency of this relation is not clear. We cannot say for sure which group is more fault-dense than the other as the directions in the figures contradict each other.

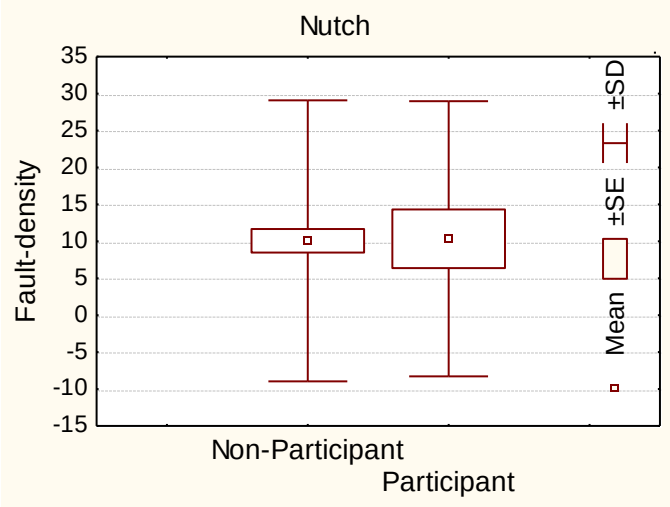
Table 5.4: P-values of Mann-Whitney U test analysis for fault-density evaluation of Participant vs. non-participant classes

	Participant vs. Non-participant
JHotDraw	0.057677
JUnit	0.386525
Lexi	**0.011515**
Nutch	0.788716
PDM	0.971235
All systems	**0.000006**




JUnit
80
60
40
20
0
-20
-40
-60
Fault-density
±SD
±SE
Mean
Non-participant
Participant


Lexi
18
16
14
12
10
8
6
4
2
0
-2
-4
-6
Fault-density
±SD
±SE
Mean
Non-participant
Participant


Nutch
35
30
25
20
15
10
5
0
-5
-10
-15
Fault-density
±SD
±SE
Mean
Non-Participant
Participant

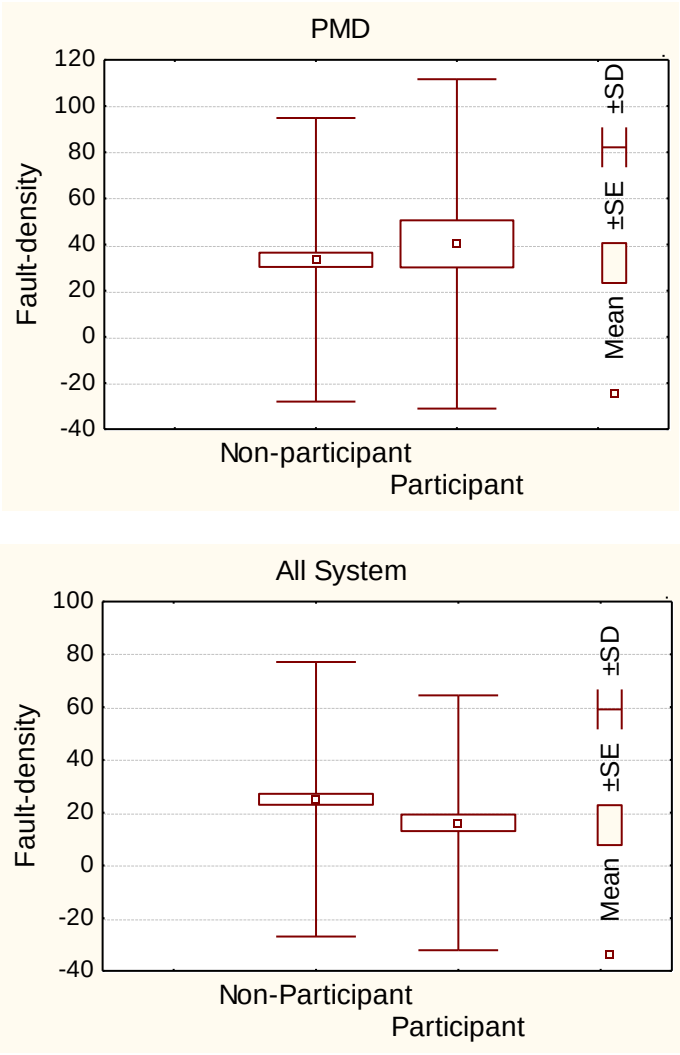


Figure 5.3: Fault-denity comparison of participant vs. non-participant classes

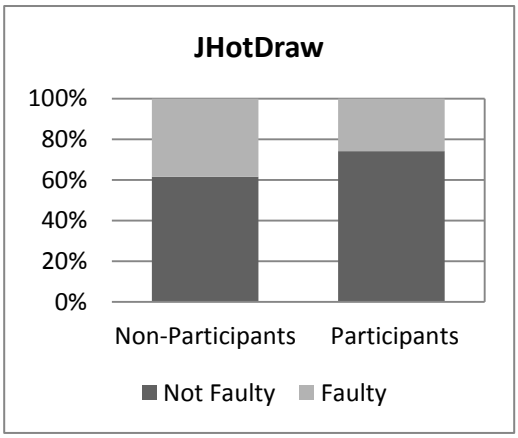
- **Fault-proneness**

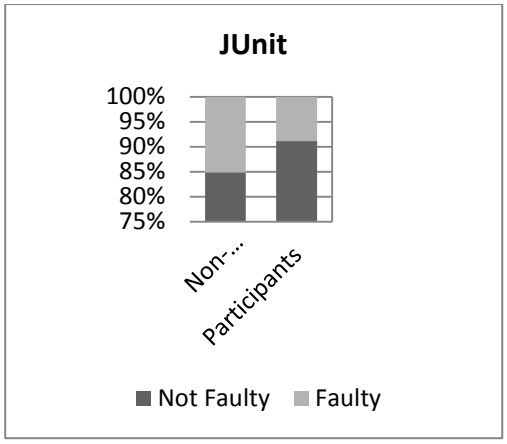
The results of evaluation of fault-proneness of participating versus non-participating classes are shown in table 5.5. Only two values show significant differences: the value associated with Lexi and the obtained value when all systems combined. We can see from figure 5.4 that the participating classes group is more fault-prone than the non-participating group in Lexi and less fault-prone than the non-participating group when all systems combined together. As it can be seen in figure 5.4, it is difficult to say which

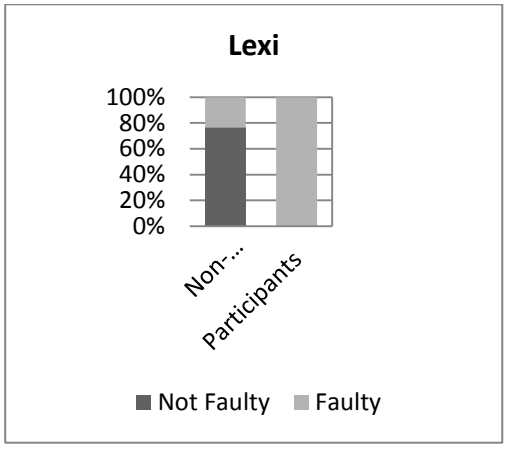
group is more fault-prone than the other since that the directions in the different figures contradicts each other.

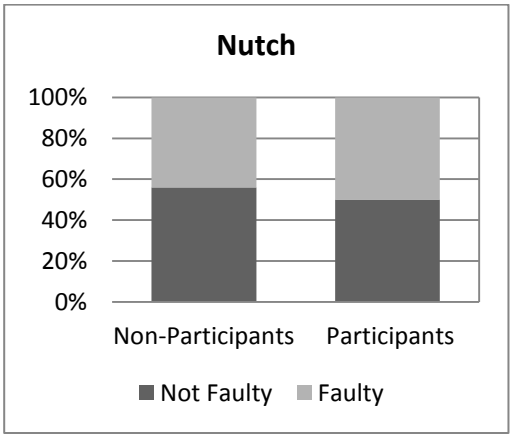
Table 5.5: P-values of Mann-Whitney U test analysis for fault-proneness evaluation of Participant vs. non-participant classes groups

	Participant vs. non-participant
JHotDraw	0.134978
JUnit	0.395434
Lexi	**0.000821**
Nutch	0.602860
PDM	0.830092
All systems	**0.000080**




JUnit
100%
95%
90%
85%
80%
75%
Non-...
Participants
Not Faulty
Faulty


Lexi
100%
80%
60%
40%
20%
0%
Non-...
Participants
Not Faulty
Faulty


Nutch
100%
80%
60%
40%
20%
0%
Non-Participants
Participants
Not Faulty
Faulty

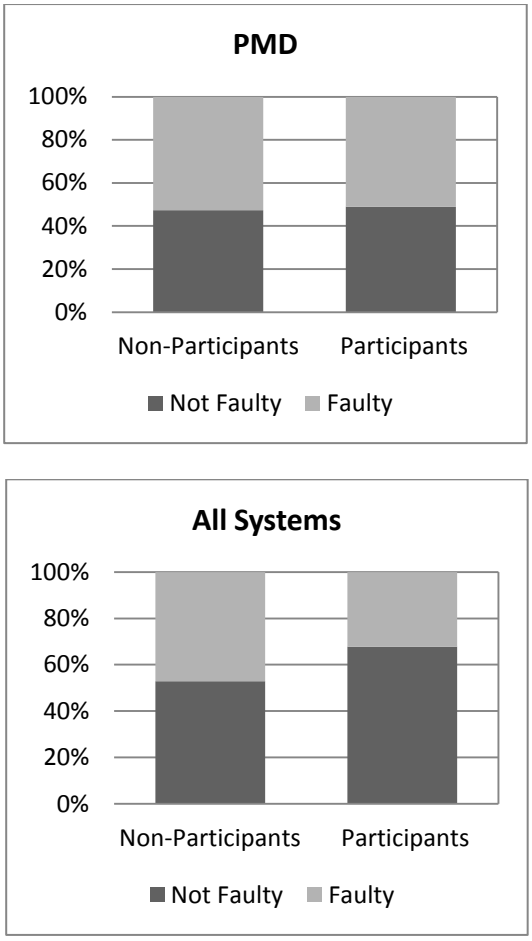


Figure 5.4: Fault-proneness comparison of participant vs. non-participant classes

5.1.2 Category Level

As mentioned in chapter 1, there are three categories for design patterns – Creational, Structural and Behavioral. The objective of this section is to evaluate the differences in modularity (coupling and cohesion) and functional correctness (fault-density and fault-proneness) of the classes that participate in the different categories of design patterns. To do so, we identified 6 pairs of categories to be evaluated in each subject system and when all systems combined together. These pairs are as follows:

1. Creational vs. Non-participant: in evaluating the differences in modularity and functional correctness of this pair, we evaluate the difference between the classes that participate in the Creational design patterns and the classes that do not participate in any design pattern.
2. Structural vs. Non-participant: in evaluating the differences in modularity and functional correctness of this pair, we evaluate the difference between the classes that participate in the Structural design patterns and the classes that do not participate in any design pattern.
3. Behavioral vs. Non-participant: in evaluating the differences in modularity and functional correctness of this pair, we evaluate the difference between the classes that participate in the Behavioral design patterns and the classes that do not participate in any design pattern.
4. Creational vs. Structural: in evaluating the differences in modularity and functional correctness of this pair, we evaluate the difference between the classes that participate in the Creational design patterns and the classes that participate in the Structural design patterns.
5. Creational vs. Behavioral: in evaluating the differences in modularity and functional correctness of this pair, we evaluate the difference between the classes that participate in the Creational design patterns and the classes that participate in the Behavioral design patterns.
6. Structural vs. Behavioral: in evaluating the differences in modularity and functional correctness of this pair, we evaluate the difference between the classes that participate in the Structural design patterns and the classes that participate in the Behavioral design patterns.

5.1.2.1 Creational vs. Non-participant

In evaluating the difference in modularity and functional correctness between the classes that participate in the creational design patterns and the non-participant classes, we ignore the Nutch and the JUnit subject systems. This is because they are only associated with 1 and 2 classes respectively.

- **Coupling**

The obtained results show that there are differences between the coupling of the non-participant classes versus the classes that participate in creational design patterns in four cases - three different systems and when all systems combined together. The p-values associated with the evaluation of coupling of the non-participant classes versus the classes that participate in the creational design patterns are presented in table 5.6. We can see that JHotDraw, Lexi and PMD are associated with the significant p-values. The same thing can be said when we combined all systems together. At the same time, we can see from figure 5.5 that in all of these cases the classes that participate in the creational design patterns are more coupled than the non-participant classes.

- **Cohesion**

Evaluating the difference in cohesion between the classes that participate in the creational design patterns and the non-participant classes results in three significant p-values as it can be seen in table 5.7. These values are associated with JHotDraw, Lexi and when all systems combined together.

As it can be seen in figure 5.6 that the figures associated with the significant p-values show that the non-participant classes are more cohesive than classes that participate in the

creational design patterns. The deviance is associated with PMD but this system is not associated with significant p-value. So, the claim that the non-participant classes are more cohesive than the classes that participate in the creational design pattern has more support.

- **Fault-density**

Two significant p-values in different directions are obtained in evaluating the fault-density of the non-participating classes versus the classes that participate in the creational design patterns. These values are associated with JHotDraw and Lexi as in table 5.8. As it can be seen in figure 5.6, the non-participant classes are more fault-dense than the classes that participate in the creational design patterns in JHotDraw and less fault-dense than the classes that participate in the creational design patterns in Lexi .

The tendency of the fault-density for the non-participant classes versus creational classes is not clear. In addition to the different directions for the significant p-values associated with JHotDraw and Lexi, we can see that the non-participant classes in PMD are less fault-dense than the classes that participate in the creational design patterns and more fault-dense when all systems combined together.

- **Fault-proneness**

The only significant difference in evaluating the fault-proneness of the non-participating classes versus the classes that participate in the creational design patterns is in Lexi as shown in table 5.9. Figure 5.8 shows that the classes that participate in the creational design patterns are more fault-prone than the non-participant classes in Lexi.

We also can see in figure 5.8 that the non-participant classes are more fault-prone than the classes that participate in the creational design patterns in JHotDraw and when combining all systems together and less fault-prone in PMD.

We can conclude that there is no clear tendency for the difference in fault-proneness of these groups. We cannot say which group is more fault-prone than the other.

5.1.2.2 Structural vs. Non-participant

We ignore the Lexi subject system in evaluating the difference in modularity and functional correctness of the structural design patterns versus the non-participant classes. This is because the number of classes in the structural category is 0.

- **Coupling**

Evaluation the coupling of the non-participant classes versus the classes that participate in the structural design patterns results in one significant p-value as it can be seen in table 5.6. This value is associated with JHotDraw. In all the other cases the p-values associated with the different cases are not significant.

We can see from figure 5.5 that the structural design patterns are more coupled than the non-participant in case of JHotDraw, Nutch, PMD and when we combined all systems. The only anomaly is JUnit. In JUnit, the non-participant classes are more coupled than the classes that participate in structural design patterns. However, the p-value associated with JUnit is insignificant.

To sum up, the obtained results suggests that the classes that participate in the structural design patterns are more coupled than the non-participant classes.

- **Cohesion**

The p-values obtained in evaluating the difference between the cohesion of the non-participating classes versus the classes that participate in the structural design patterns are shown in table 5.7. As it can be seen in table 5.7, there are three significant p-values. These values associated with JHotDraw, PMD and when all systems combined together.

We can see from figure 5.6 that the non-participant classes are more cohesive than the classes that participate in the structural design patterns in all cases except for JUnit which is associated with an insignificant p-value.

We can conclude that the non-participant classes tend to be more cohesive than the classes that participate in the structural design pattern.

- **Fault-density**

One significant p-value is obtained in evaluating the fault-density of non-participant classes versus the classes that participate in structural design patterns as shown in table 5.8. This value is obtained when we combined all systems together. As it can be seen in figure 5.7, the non-participant classes are more fault-dense than the classes that participate in structural design patterns.

For JHotDraw and JUnit, they are not significant at 0.05 level. However, they are significant at 0.1 level as it can be seen in table 5.8. They also support the results obtained when all systems combined together.

The same thing can be said about the tendency of fault-density in case of Nutch and PMD, even though the p-value associated them are not significant. As it can be seen in figure

5.7, the non-participant classes are more fault-dense than the classes that participate in the structural design patterns as well.

So, it is clear that the non-participant classes tend to be more fault-dense than the classes that participate in the structural design patterns.

- **Fault-proneness**

The evaluation of fault-proneness of the non-participating classes and the classes that participate in the structural design patterns results in one significant p-value. This value is obtained when all systems combined together as shown in table 5.9. As it can be seen in figure 5.8 that the non-participating classes are more fault-prone than the classes that participate in structural design patterns.

In all the other cases, the p-values were insignificant but in all of these cases the non-participating classes are more fault-prone than all the classes that participating in the structural design patterns.

In this case we can easily infer that classes that participate in the structural design patterns are less fault-prone than non-participant classes.

5.1.2.3 Behavioral vs. Non-participant

- **Coupling**

The results of evaluating the difference in coupling between the classes that participate in the behavioral design patterns and the non-participant classes are significant in all cases except for the JUnit as it can be seen in table 5.6.

By examining these differences, we can see that the behavioral design patterns are more coupled than the non-participant classes in all of the cases even in case of JUnit as it can be seen in figure 5.5.

- **Cohesion**

The p-values associated with the evaluation the cohesion of the non-participant classes versus the classes that participate in the behavioral design patterns are significant in all cases except in JUnit as it can be seen in table 5.7.

Regarding the tendency of the relationship, we can see in figure 5.6 that the non-participant classes are more cohesive than the classes that participate in the behavioral design pattern in all cases.

- **Fault-density**

Evaluation the fault-density of the non-participant classes versus the classes that participate in behavioral design patterns results in two significant p-values in two directions. The first value is associated with Lexi and the second value is obtained when all systems are combined together as in table 5.8. As it can be seen in figure 5.7, the non-participant are less fault-dense than the classes that participate in behavioral design patterns in Lexi and more fault-dense than the classes that participate in behavioral design patterns when all systems are combined together.

It is difficult to detect which tends to be more fault-dense: the non-participant classes or the classes that participate in the behavioral design patterns. This is because the conclusion drawn from the JHotDraw, PMD, JUnit and when combining all systems

together is in one direction and the conclusion drawn from Lexi and Nutch is in the other direction as it can be seen in figure 5.7.

- **Fault-proneness**

There are two significant differences in two different directions associated with evaluating the fault-proneness of the classes that participate in the behavioral design patterns versus the non-participant classes. The first one is associated with Lexi and the second is when all systems combined together as in table 5.9. In Lexi, the classes that participate in the behavioral design patterns are more fault-prone than the non-participating classes as it can be seen in figure 5.8. But this is not the case when we combined all systems. When all systems are combined, the classes that participate in the behavioral design patterns are less fault-prone than the non-participating classes.

The other p-values are insignificant and in different directions. The non-participant classes are more fault-prone than the classes that participate in the behavioral design patterns in JHotDraw and PMD and less fault-prone than the classes that participate in the behavioral design patterns in Nutch and JUnit.

There is no clear tendency for this relationship. It is difficult to say which group tends to be more fault-prone than the other.

5.1.2.4 Creational vs. Structural

In evaluating the difference in modularity and functional correctness between the classes that participate in the creational design patterns and the classes that participate in the structural design patterns, we ignore the Nutch and the JUnit subject systems. This is because they are only associated with 1 and 2 classes respectively. Also, we ignore Lexi

because the number of the classes that participate in the structural design patterns is 0. So, we end up with three cases only – JHotDraw, PMD and the case when all systems combined.

- **Coupling**

In comparing the coupling of the classes that participate in the creational design patterns versus the classes that participate in the structural design patterns, only one significant p-value is obtained. This value is associated with the JHotDraw as it can be seen in table 5.6. In all the other addressed cases the obtained value are insignificant. However, the classes that participate in the structural design patterns seems to be more coupled than the classes that participate in the creational design patterns in all cases as it can be seen in figure 5.5.

We can see that there is a general tendency for the relationship. We can see that the classes that participate in the Structural design patterns are more coupled than the classes that participate in the creational design patterns.

- **Cohesion**

The only significant p-value in evaluating the difference in cohesion between the classes that participate in the creational design patterns and the structural design patterns is associated with PMD as it can be seen in table 5.7.

We can see that it is difficult to infer the tendency of the relationship between the classes that participate in the creational design patterns and the classes that participate in behavioral design patterns. As it can be seen in figure 5.6 that in some cases the classes

that participate in the creational design patterns tend to be more cohesive and in other cases the structural design patterns tend to be more cohesive.

- **Fault-density**

Only one significant p-value is obtained as a result of evaluating fault-density of the classes that participate in creational design patterns versus the classes that participate in structural design patterns. This value is obtained when all systems combined as in figure 5.7. In this case the classes that participate in the creational design patterns are more fault-dense than the classes that participate in the structural design patterns.

This finding is supported by JHotDraw and PMD. Although the p-values associated with JHotDraw and PMD are not significant but they provide more support for the results obtained when combining all systems together.

We can see that the classes that participate in the creational design patterns tend to be more fault-dense than the classes that participate in the structural design patterns.

- **Fault-proneness**

The p-values obtained in evaluating the difference in fault-proneness between the classes that participate in the creational design patterns and the classes that participate in the structural design patterns are significant in one case only - when all systems combined as shown in table 5.9. In this case the classes that participate in creational design patterns are more fault-prone than the classes that participate in structural design patterns as it can be seen in figure 5.8.

The p-values associated with JHotDraw and PMD are insignificant and inconsistent. The classes that participate in creational design patterns are more fault-prone than the classes that participate in structural design patterns in PMD and are less fault-prone than the classes that participate in structural design pattern in JHotdraw.

Considering that the PMD is of a bigger size compared to the other systems. If we also take into account that the reported difference in case of JHotDraw is small. We can see that the classes that participate in the creational design patterns tends to be more fault-prone than the classes that participate in structural design patterns.

5.1.2.5 Creational vs. Behavioral

In evaluating the difference in modularity and functional correctness between the classes that participate in the creational design patterns and the classes that participate in the behavioral design patterns, we ignore the Nutch and the JUnit subject systems. This is because they are only associated with 1 and 2 classes respectively

- **Coupling**

The evaluation of coupling of the classes that participate in the creational design patterns versus the classes that participate in the behavioral design patterns results in insignificant p-values in all cases as it can be seen in table 5.6.

For the general tendency of this relationship, as it can be seen in figure 5.5 that in all cases, except for Lexi, the classes that participate in the behavioral design patterns tends to be more coupled than the classes that participate in the creational design patterns. For Lexi, it has small number of classes so it has a low impact on the general conclusion.

However, we cannot adopt this conclusion in our work because it is not supported with significant p-values.

- **Cohesion**

It can be seen in table 5.7 that only one significant p-value is reported in evaluating the difference between the cohesion of the classes that participate in the creational design patterns and the cohesion of the classes that participate in the behavioral design patterns. This value is associated with PMD.

The tendency of this relationship is not clear. As it can be seen in figure 5.6 that in some cases the classes that participate in the creational design pattern are more cohesive but in the other cases the behavioral design patterns are more cohesive. For PMD and when all systems combined together, the classes that participate in the creational design patterns tends to be more cohesive than the behavioral design patterns. But it is the other way around for JHotDraw. For Lexi, we can see that both groups are the same.

- **Fault-density**

Two significant p-values in two different directions are reported in evaluation the fault-density of the classes that participate in creational design patterns and the classes that participate in behavioral design patterns as in table 5.8. These values are associated with JHotDraw and PMD. In JHotDraw, the classes that participate in creational design patterns are less fault-dense than the classes that participate in behavioral design patterns as in figure 5.7. In PMD, the classes that participate in the creational design patterns are more fault-dense than the classes that participate in behavioral design patterns.

It is difficult to say which tends to be more fault-dense: the classes that participate in creational design patterns or the classes that participate in behavioral design patterns. This is because the differences in JHotDraw and Lexi are in one direction and in PMD and when all systems combined are in the other direction.

- **Fault-proneness**

The evaluation of fault-proneness in the classes that participate in the creational design patterns versus the classes that participate in the behavioral design patterns results in one significant p-value. This value is associated with JHotDraw as it can be seen in table 5.9. Figure 5.8 shows that the classes that participate in behavioral design patterns are more fault-prone than the classes that participate in the creational design patterns.

All the other systems are associated with insignificant p-values and it is hard to detect the general tendency for the results associated with these systems. As it can be seen in figure 5.8, the classes that participate in the creational design patterns in PMD and when all systems combined together are more fault-prone than the classes that participate in the behavioral design patterns. For Lexi, all the classes that participate in both categories are faulty so it is not differentiating.

5.1.2.6 Structural vs. Behavioral

We ignore the Lexi subject system in evaluating the difference in modularity and functional correctness of the structural design patterns versus the classes that participate in the behavioral design patterns. This is because the number of classes in the structural category is 0.

- **Coupling**

The relationship between the coupling of the classes that participate in the structural design patterns and the classes that participate in the behavioral design patterns is evaluated on the subject systems and the results presented in table 5.6. The obtained p-values are significant in three cases – Nutch, PMD and when all systems combined together.

At the same time, it is hard to tell which group is more coupled: the classes that participate in the structural design patterns or the classes that participate in the behavioral design patterns. We can see form figure 5.5 that the classes that participate in the behavioral design patterns are more coupled than the classes that participate in the structural design patterns in JUnit, Nutch and when all the systems combined together. But it is the other way around in case of JHotDraw and to some extent in PMD.

- **Cohesion**

The relationship between the cohesion of the classes that participate in the structural design patterns and the cohesion of the classes that participate in the behavioral design patterns is not clear. We can see in table 5.7 that only one significant p-value is obtained. This value is reported when all systems combined together.

As it can be seen in figure 5.6 that some cases reported that the classes that participate in the structural design patterns are more cohesive while other cases reported that the classes that participate in the behavioral design patterns are more cohesive. The classes that participate in the structural design patterns are more cohesive than the classes that participate in the behavioral design patterns in JUnit and when all systems combined together. But it is the other way around in the other cases.

- **Fault-density**

Two significant p-values in the same direction are obtained in evaluation the fault-density of the classes that participate in the structural design patterns versus the classes that participate in the behavioral design patterns as we can see table 5.8. These values associated with Nutch and when all systems combined together. In both of these cases the classes that participate in the structural design patterns are less fault-dense than the classes that participate in behavioral design patterns as in figure 5.7.

The p-value associated with JHotDraw is insignificant though it provides further support for the conclusion drawn above. For PMD, the result obtained is in the opposite direction but it is not significant as it can be seen in figure 5.7. As in figure 5.7, the results obtained in evaluating fault-density for the classes that participate in both categories in JUnit are the same.

We can say that the classes that participate in the structural design patterns are less fault-dense than the classes that participate in behavioral design patterns.

- **Fault-proneness**

Only one significant p-value is obtained when the fault-proneness of the classes that participates in the structural design patterns versus the classes that participate in behavioral design patterns is evaluated. This value is obtained when we combined all systems as in table 5.9. From figure 5.8, we can see that the classes participate in behavioral design patterns are more fault-prone than the classes participate structural design patterns.

All the other p-values are not significant but we can detect a general tendency. We can see from figure 4 that the classes that participate in the behavioral design patterns are also more fault-prone than the classes structural design patterns in all of cases except in case of PMD.

Table 5.6: P-values associated with evaluating coupling of the different categories

JHotDraw	Creational	Structural	Behavioral
Non-Participant	**0.005938**	**0.000481**	**0.007307**
Creational	-	**0.027462**	0.676136
Structural	-	-	0.238172

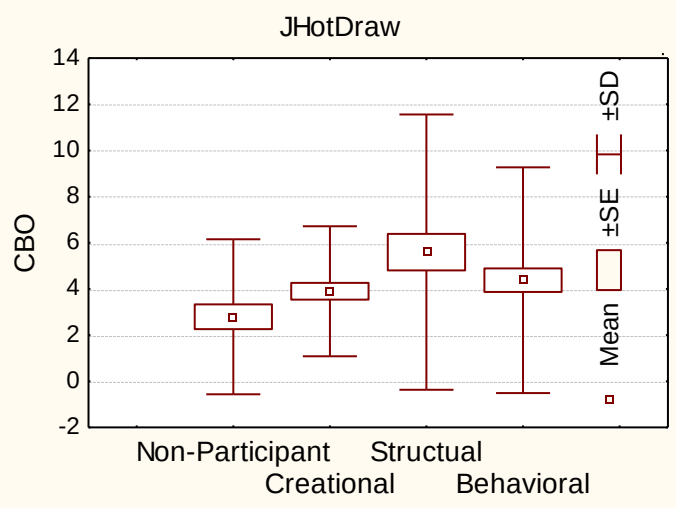
JUnit	Creational	Structural	Behavioral
Non-Participant	0.090737	0.591811	0.131729
Creational	-	0.108451	0.090603
Structural	-	-	0.070657

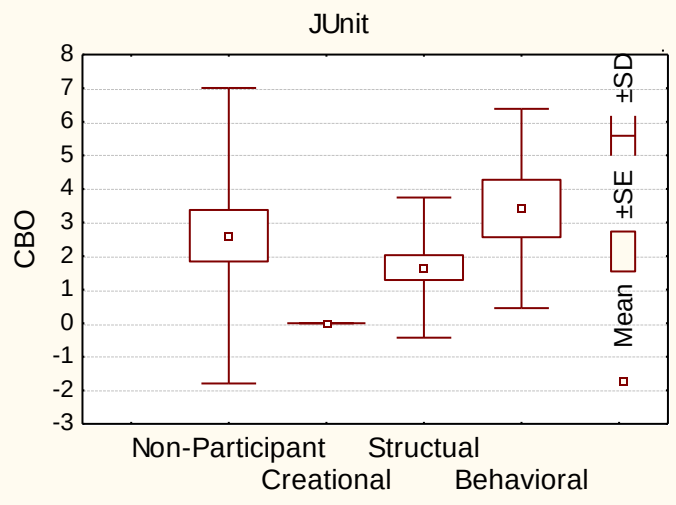
Lexi	Creational	Structural	Behavioral
Non-Participant	**0.000299**	1.000000	**0.015730**
Creational	-	1.000000	1.000000
Structural	-	-	1.000000

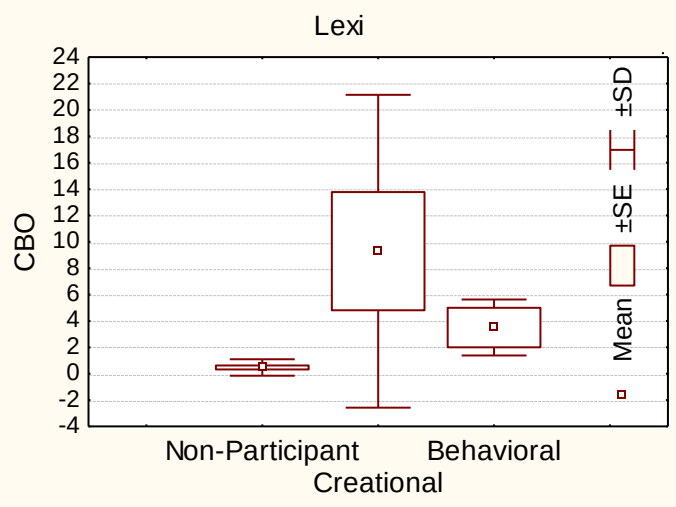
Nutch	Creational	Structural	Behavioral
Non-Participant	1.000000	0.096719	**0.000033**
Creational	-	1.000000	1.000000
Structural	-	-	**0.033273**

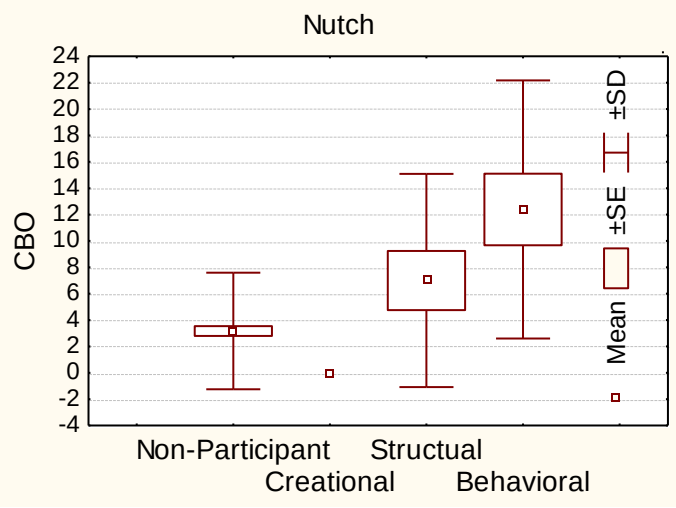
PDM	Creational	Structural	Behavioral
Non-Participant	**0.032606**	0.447532	**0.018900**
Creational	-	0.203568	0.348754
Structural	-	-	**0.022461**

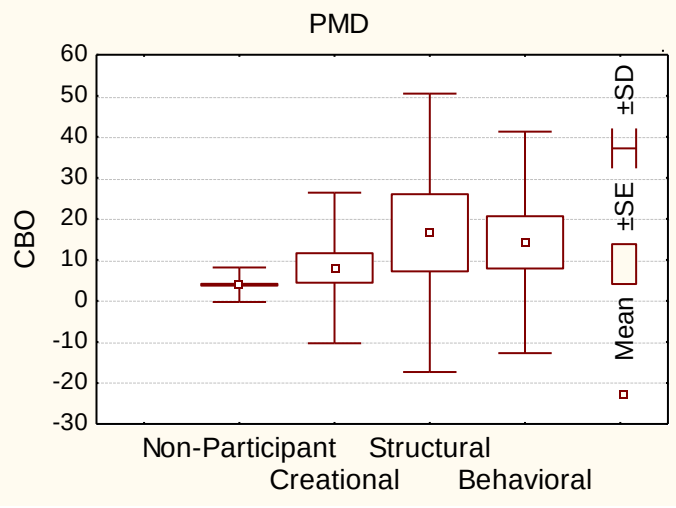
All systems	Creational	Structural	Behavioral
Non-Participant	**0.002779**	0.302712	**0.000095**
Creational	-	0.128179	0.436638
Structural	-	-	**0.001915**






Lexi
CBO
24
22
20
18
16
14
12
10
8
6
4
2
0
-2
-4
Non-Participant
Behavioral
Creational
±SD
±SE
Mean


Nutch
CBO
24
22
20
18
16
14
12
10
8
6
4
2
0
-2
-4
Non-Participant
Structual
Creational
Behavioral
±SD
±SE
Mean


PMD
CBO
60
50
40
30
20
10
0
-10
-20
-30
Non-Participant
Structural
Creational
Behavioral
±SD
±SE
Mean

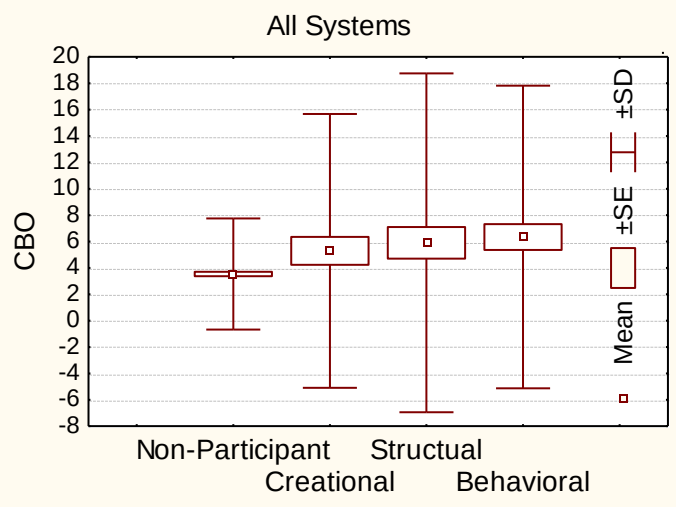


Figure 5.5: Coupling comparison for the different categories

Table 5.7: P-values associated with evaluating cohesion of the different categories

JHotDraw	Creational	Structural	Behavioral
Non-Participant	**0.001696**	**0.009208**	**0.025845**
Creational	-	0.216010	0.704056
Structural	-	-	0.471754

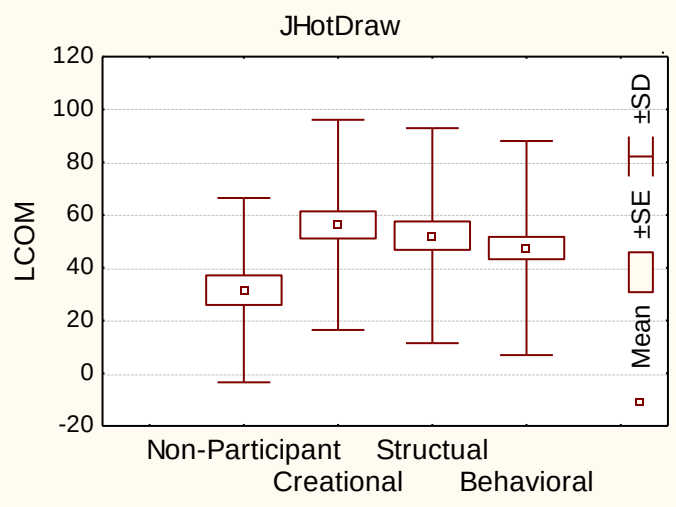
JUnit	Creational	Structural	Behavioral
Non-Participant	0.214485	0.091484	0.374218
Creational	-	0.332527	0.171791
Structural		-	0.053273

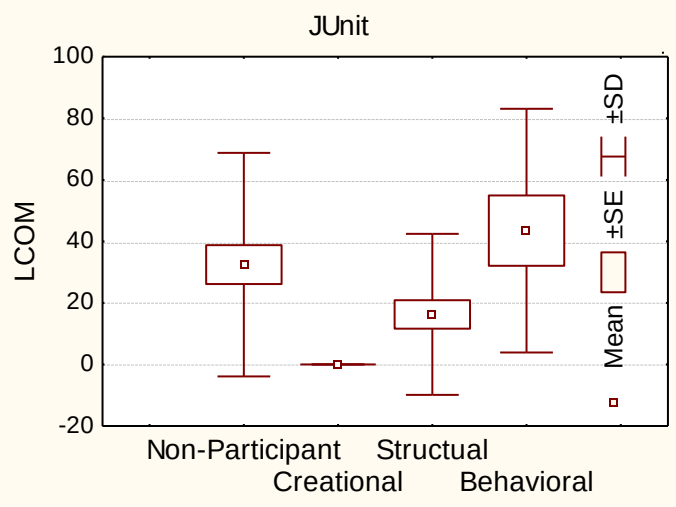
Lexi	Creational	Structural	Behavioral
Non-Participant	**0.000484**	1.000000	**0.036662**
Creational	-	1.000000	1.000000
Structural	-	-	1.000000

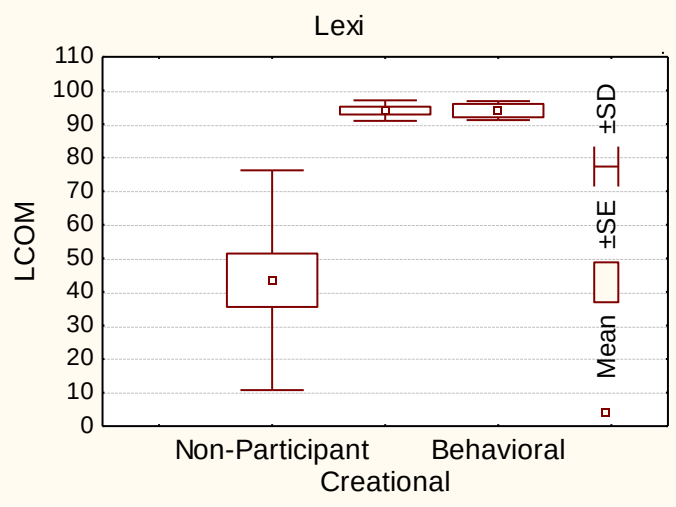
Nutch	Creational	Structural	Behavioral
Non-Participant	1.000000	0.479091	**0.022671**
Creational	-	1.000000	1.000000
Structural	-	-	0.138253

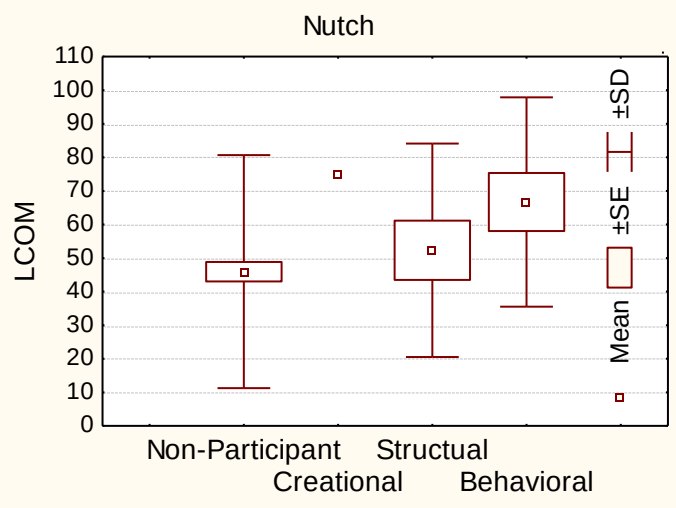
PMD	Creational	Structural	Behavioral
Non-Participant	0.314681	**0.032995**	**0.016246**
Creational	-	**0.018336**	**0.021568**
Structural	-	-	0.265996

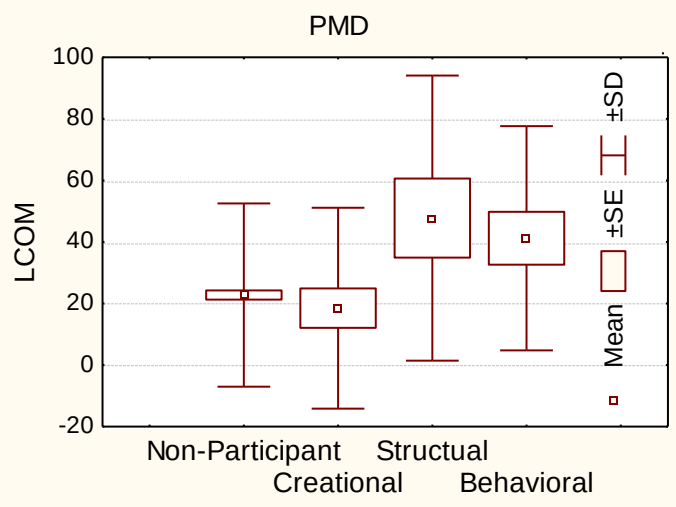
All system	Creational	Structural	Behavioral
Non-Participant	**0.000014**	**0.001234**	**0.000000**
Creational	-	0.190215	0.110084
Structural	-	-	**0.019132**






Lexi
LCOM
110
100
90
80
70
60
50
40
30
20
10
0
±SD
±SE
Mean
Non-Participant
Behavioral
Creational


Nutch
LCOM
110
100
90
80
70
60
50
40
30
20
10
0
±SD
±SE
Mean
Non-Participant
Structual
Creational
Behavioral


PMD
LCOM
100
80
60
40
20
0
-20
±SD
±SE
Mean
Non-Participant
Structual
Creational
Behavioral

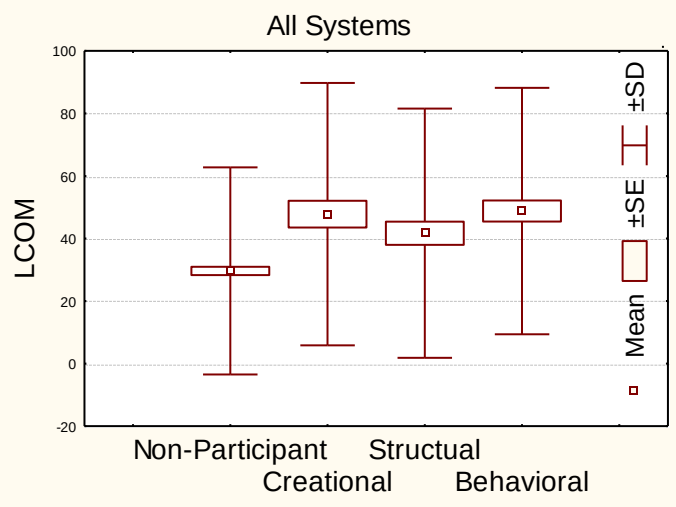


Figure 5.6: Cohesion comparison of the different categories.

Table 5.8: P-values associated with evaluating fault-desnity of the different categories

JHotDraw	Creational	Structural	Behavioral
Non-Participant	**0.040954**	0.074557	0.190715
Creational	-	0.623715	**0.010846**
Structural		-	0.089498

JUnit	Creational	Structural	Behavioral
Non-Participant	0.129812	0.094472	1.000000
Creational	-	**0.005415**	0.204005
Structural		-	0.116530

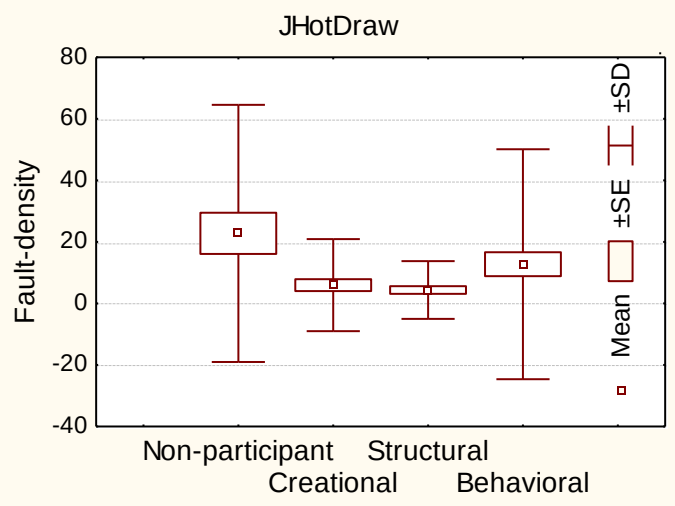
Lexi	Creational	Structural	Behavioral
Non-Participant	**0.011515**	1.000000	**0.036336**
Creational	-	1.000000	1.000000
Structural		-	1.000000

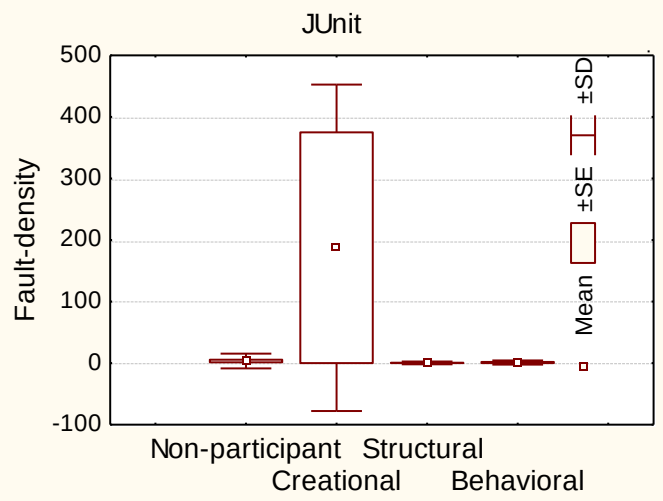
Nutch	Creational	Structural	Behavioral
Non-Participant	1.000000	0.309721	0.113882
Creational	-	1.000000	1.000000

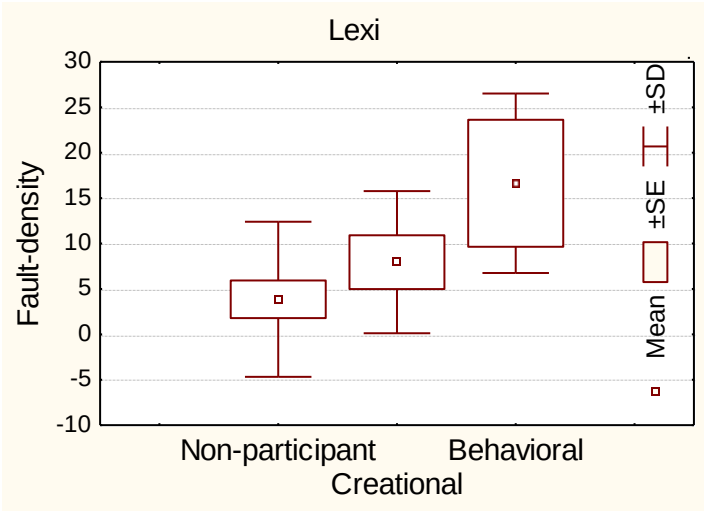
Structural		-	**0.024861**

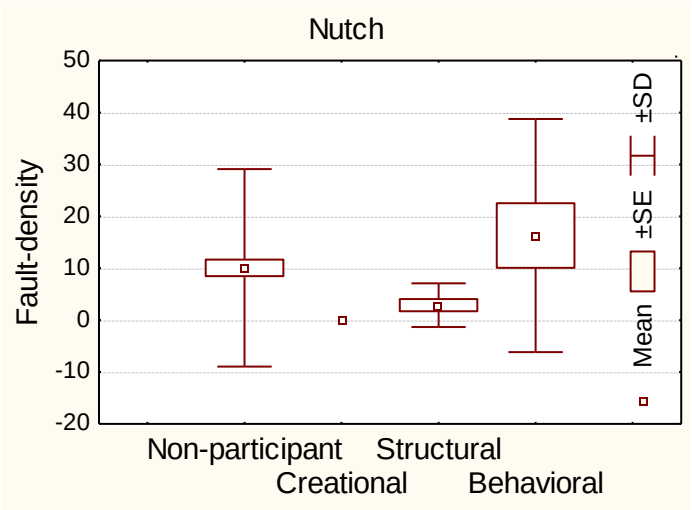
PMD	**Creational**	**Structural**	**Behavioral**
Non-Participant	0.201336	0.774087	0.091711
Creational	-	0.476195	**0.041217**
Structural		-	0.299755

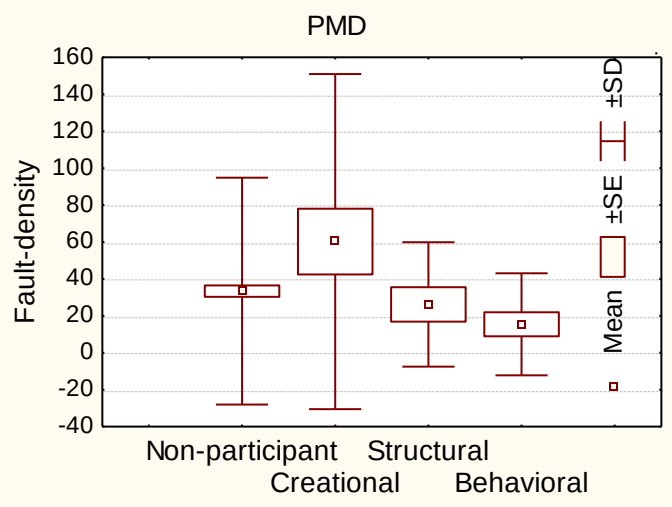
All system	**Creational**	**Structural**	**Behavioral**
Non-Participant	0.082337	**0.000000**	**0.000480**
Creational	-	**0.000135**	0.194804
Structural		-	**0.005921**






Lexi
Fault-density
30
25
20
15
10
5
0
-5
-10
±SD
±SE
Mean
Non-participant
Behavioral
Creational


Nutch
Fault-density
50
40
30
20
10
0
-10
-20
±SD
±SE
Mean
Non-participant
Structural
Creational
Behavioral


PMD
Fault-density
160
140
120
100
80
60
40
20
0
-20
-40
±SD
±SE
Mean
Non-participant
Structural
Creational
Behavioral

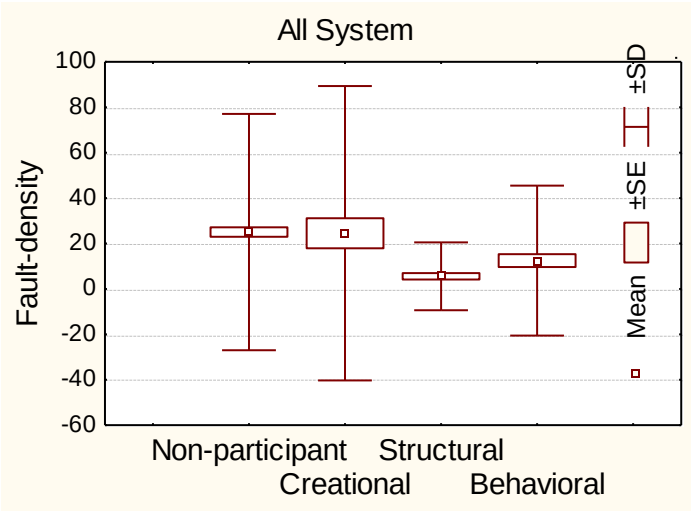


Figure 5.7: Fault-density comparison of the different categories.

Table 5.9: P-values associated with evaluating fault-proneness of the different categories

JHotDraw	Creational	Structural	Behavioral
Non-Participant	0.107857	0.287413	0.372837
Creational	-	0.227302	**0.010247**
Structural	-	-	0.127523

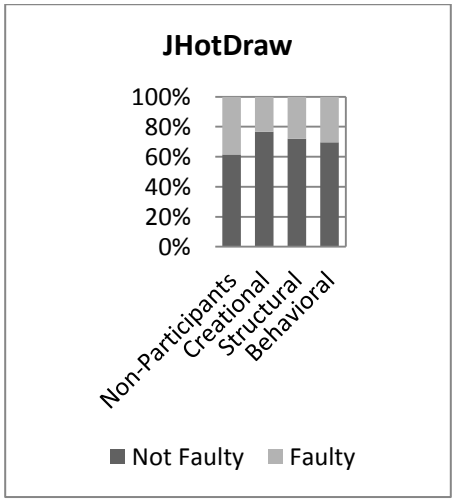
JUnit	Creational	Structural	Behavioral
Non-Participant	0.210764	0.096568	0.902402
Creational	-	**0.007087**	0.305392
Structural	-	-	0.102070

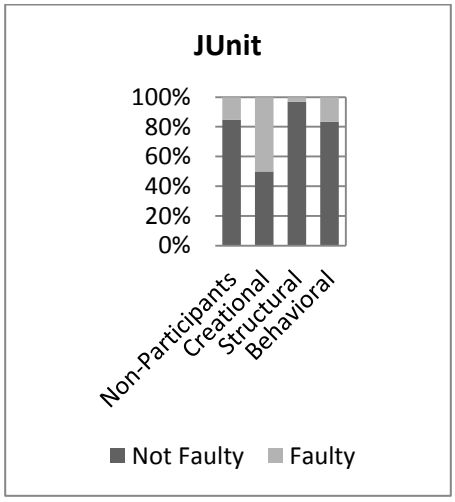
Lexi	Creational	Structural	Behavioral
Non-Participant	**0.000821**	1.000000	**0.032193**
Creational	-	1.000000	1.000000
Structural	-	-	1.000000

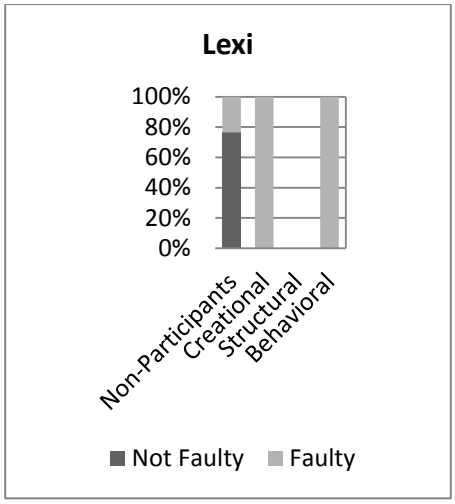
Nutch	Creational	Structural	Behavioral
Non-Participant	1.000000	0.697863	0.082271
Creational	-	1.000000	1.000000
Structural	-	-	0.052808

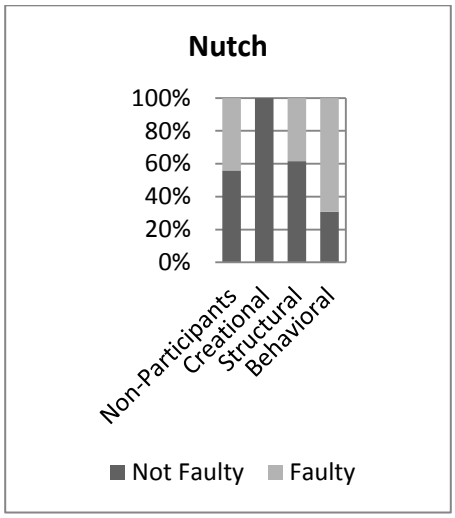
PDM	Creational	Structural	Behavioral
Non-Participant	0.379178	0.645102	0.109197
Creational	-	0.511773	0.068957
Structural	-	-	0.445526

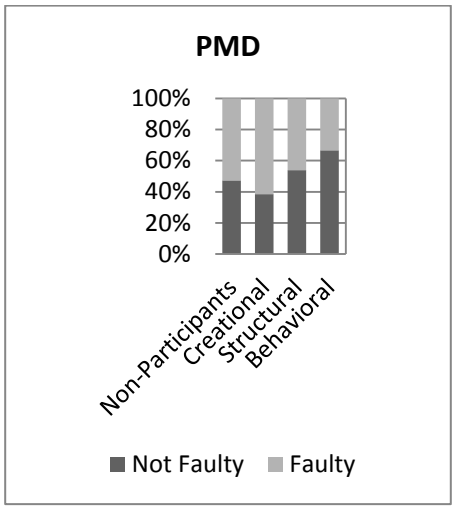
All systems	Creational	Structural	Behavioral
Non-Participant	0.171390	**0.000006**	**0.006564**
Creational	-	**0.000720**	0.321889
Structural	-	-	**0.008493**






Lexi
100%
80%
60%
40%
20%
0%
Non-Participants
Creational
Structural
Behavioral
Not Faulty
Faulty


Nutch
100%
80%
60%
40%
20%
0%
Non-Participants
Creational
Structural
Behavioral
Not Faulty
Faulty


PMD
100%
80%
60%
40%
20%
0%
Non-Participants
Creational
Structural
Behavioral
Not Faulty
Faulty

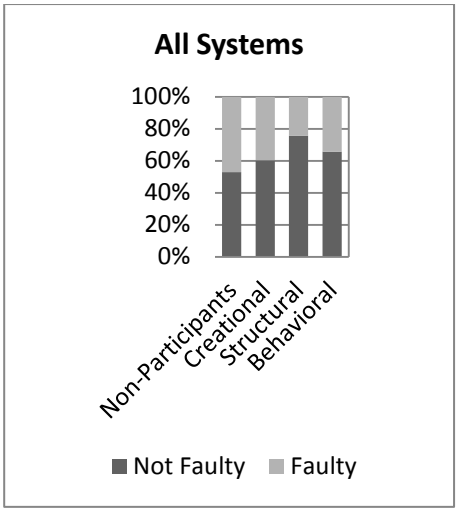


Figure 5.8: Fault-proneness comparison of the different categories

5.1.3 Pattern and Role Levels

The evaluation of modularity (coupling and cohesion) and functional correctness (fault-density and fault-proneness) of design patterns is presented in this section. In evaluating the modularity and functional correctness of each design pattern, we evaluate the following:

- The difference in modularity and functional correctness of the classes that participate in a design pattern and the classes that do not participate in that design pattern. To do so, we use Mann-Whitney test.
- Then we evaluate the overall difference among the different roles of each design pattern. To do so, we use Kruskal-Wallis test.
- If we find difference among the different roles of a design pattern, we go further to evaluate the differences between each pair of roles using Mann-Whitney test and we report the significant pairs only.

Modularity and functional correctness evaluation, of each design patterns and their roles, is conducted when all systems combined together. We do not evaluate modularity and functional correctness in each subject system. The reason is that: not each system contains the same set of patterns so we can compare.

5.1.3.1 Creational Design Patterns

5.1.3.1.1 Builder

- **Coupling and Cohesion**

Evaluation of coupling and cohesion in Builder design pattern results in insignificant p-values as it can be seen in table 5.10. This, in turns, means that there is no significant difference in the coupling and cohesion between the classes that participate in the Builder design pattern and the classes that do not participate in this pattern and among its roles.

- **Fault-density**

In evaluating the fault-density of the Builder design pattern, it was found that the Builder classes are more fault-dense than the non-Builder classes. The p-value associated with evaluation of fault-density of the Builder classes versus the non-Builder classes is significant as shown in table 5.10. We can see that in figure 5.9 that the Builder classes are more fault-dense than the non-Builder classes.

The p-value associated with evaluating fault-density of the participating classes in the different roles of the Builder design pattern is not significant as in table 5.10. As it can be seen in figure 5.9, the classes that participate in the Director roles seem to be more fault-dense than the classes that participate in the other roles. However, this difference is not significant.

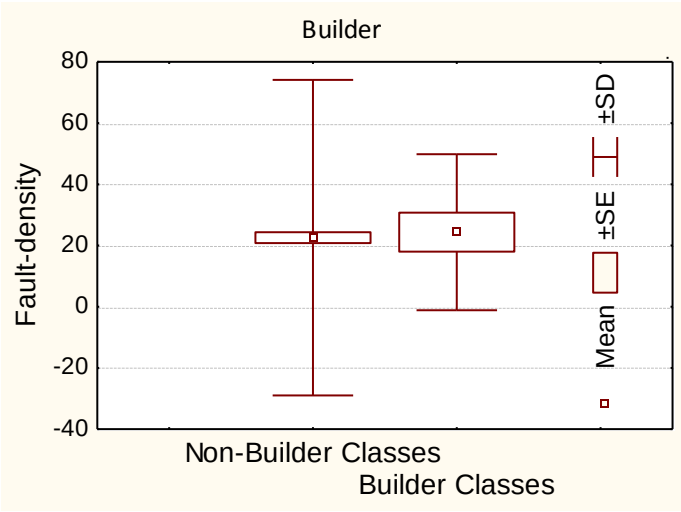
- **Fault-proneness**

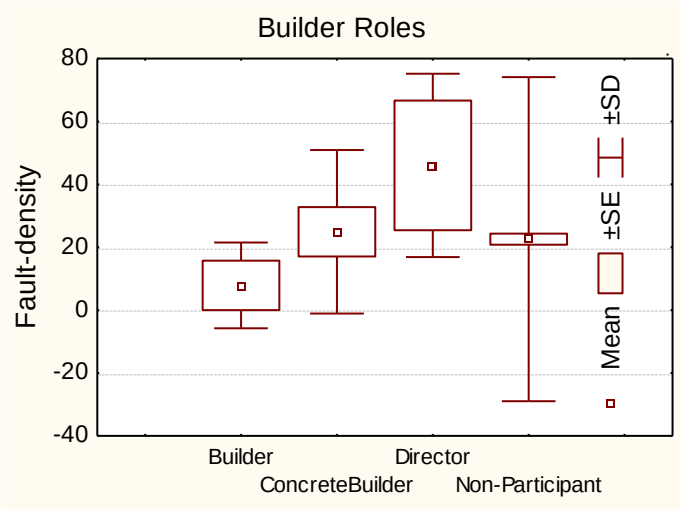
The evaluation of fault-proneness of the Builder classes versus the non- Builder classes results in significant p-value as in table 5.10. The obtained results suggest that there is a significant difference in fault-proneness between the Builder classes and the non- Builder classes. As it can be seen in figure 5.9, the classes that participate in the Builder design pattern are more fault-prone than the non-Builder classes.

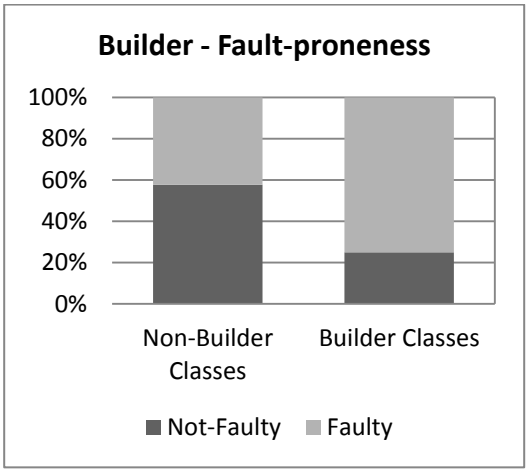
Evaluating the difference in fault-proneness among the different roles of Builder classes results in a significant difference, as we see in table 5.10. Only One pair of Builder roles results in a significant difference. This pair is: Concrete-Builder vs. Non-participant. As it can be seen in figure 5.9, the Concrete Builder classes are more fault-prone than the classes that participate in Non-participant classes.

Table 5.10: Evaluation results of Builder pattern and its roles

Builder	
CBO	
Builder classes vs. Non-Builder classes	0.123919
Overall Roles Comparison	0.194
LCOM	
Builder classes vs. Non-Builder classes	0.582115
Overall Roles Comparison	0.701
Fault-density	
Builder classes vs. Non-Builder classes	**0.043137**
Overall Roles Comparison	0.074
Fault-proneness	
Builder classes vs. Non-Builder classes	**0.009039**
Overall Roles Comparison	**0.022**
Concrete-Builder vs. Non-participant	**0.009**


Builder
80
60
40
20
0
-20
-40
Fault-density
±SD
±SE
Mean
Non-Builder Classes
Builder Classes


Builder Roles
80
60
40
20
0
-20
-40
Fault-density
±SD
±SE
Mean
Builder
ConcreteBuilder
Director
Non-Participant


Builder - Fault-proneness
100%
80%
60%
40%
20%
0%
Non-Builder Classes
Builder Classes
Not-Faulty
Faulty

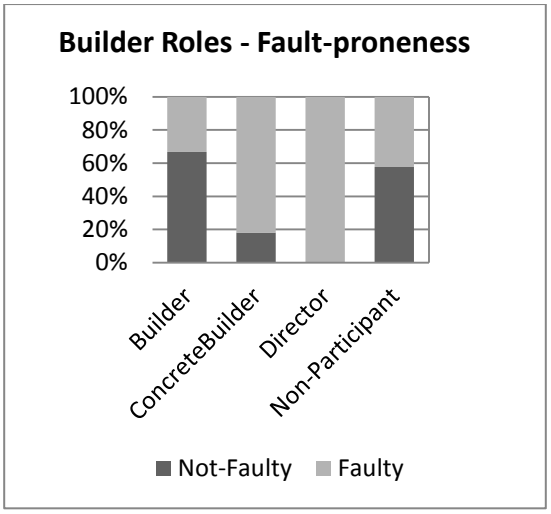


Figure 5.9: Comparison of fault-density and fault-proneness of the Builder design pattern and its roles

5.1.3.1.2 Factory Method

- **Coupling**

The p-value associated with evaluation the difference in coupling between the Factory Method classes and the non-Factory Method classes is significant. This indicates that there is significant difference in the coupling between the Factory Method classes and the non-Factory Method classes. As it can be seen in figure 5.10, the Factory Method classes are more coupled than the non-Factory Method classes.

In evaluating the differences in the classes that participate in the different roles, it was found that there is a significant difference as it can be seen in table 5.11. Three different pairs of roles show significant differences. These pairs of roles are: Product vs. Concrete-Creator, Non-participant vs. Concrete-Creator and Non-participant vs. Concrete-Product. As it can be seen in figure 5.10, the classes that participate in the Concrete-Creator and in the Concrete-Product roles are more coupled than the non-participant classes. Also, we

can see that the classes that participate in the Product role are more coupled than the classes that participate in the Concrete-Creator role.

- **Cohesion**

The p-value associated with the evaluating the difference in cohesion between the classes that participate in the Factory Method classes and the non-Factory Method classes is significant as shown in table 5.11. The obtained p-value indicates that there is significant difference between the cohesion of the Factory Method classes and the non-Factory classes. As it can be seen in figure 5.11, the non-Factory Method classes are more cohesive than the Factory Method classes.

The evaluation of the differences among the classes that participate in the different roles of the Factory Method results in significant differences. It can be seen in table 5.11 that the p-values associated with evaluating the differences among the Factory Method roles are significant. Four pairs of roles show significant differences. These roles are: Creator vs. Concrete-Creator, Product vs. Concrete-Creator, Non-participant vs. Concrete-Creator and Concrete-Product vs. Concrete-Creator. As it can be seen in figure 5.10 that the classes that participate in the Concrete-Creator role are less cohesive than the classes that participate in the Creator, Product and Concrete-Product roles and less cohesive than the non-participant classes.

- **Fault-density**

There is a significant difference in the fault-density between the Factory Method classes and the non-Factory Method classes as we can see in the associated p-value in table 5.11.

As it can be seen in figure 5.10, the Factory Method classes are less fault-dense than the non-Factory Method classes.

The evaluation of the difference in fault-proneness among the classes that participate in the different roles of Factory Method results in significant p-value as it can be seen in table 5.11. This indicates that there is significant difference among the different roles of the Factory Method design pattern. Only one pair of roles shows a significant difference. This pair is: Concrete-Product vs. Non-participant. As it can be seen in figure 5.10, the classes that participate in the Concrete-Product role are more fault-dense than the non-participant classes.

- **Fault-proneness**

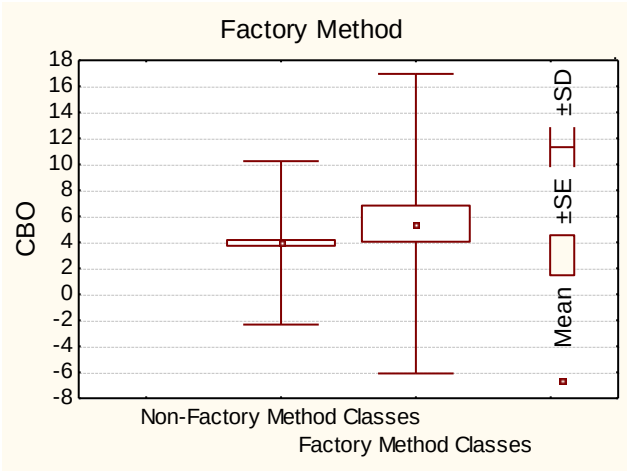
The p-value associated with evaluating the difference in fault-proneness between the Factory Method classes and the non-Factory Method classes is significant as in table 5.10. It indicates that there is a significant difference in the fault-proneness of the non-Factory Method classes and the fault-proneness of the non-Factory Method classes. As it can be seen in figure 5.10, the Factory Method classes are less fault-prone than the non-Factory Method classes.

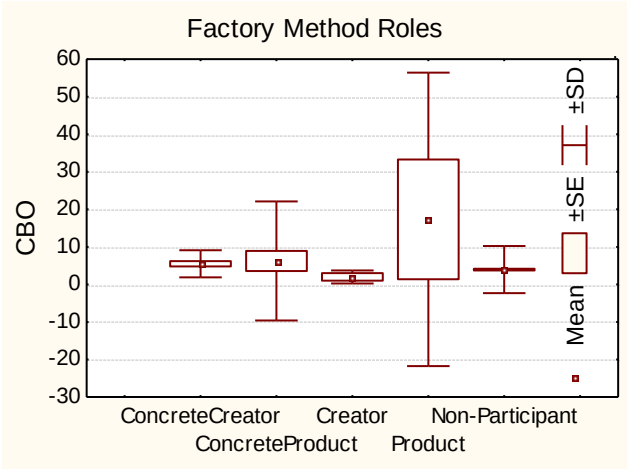
In evaluating the difference in fault-proneness among the different roles of Factory Method classes, it was found that there is significant difference in the fault-proneness among the classes that participate in the different roles of Factory Method design pattern as we can see in the associated p-value. Only two pairs of roles results in significant differences. These pairs are: Concrete-Product vs. Non-participant and Concrete-Product vs. Concrete-Creator. As it can be seen in figure 5.10, the classes that participate in the

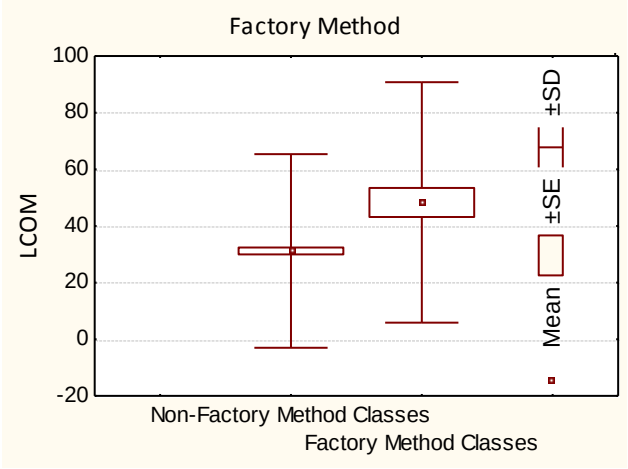
Concrete-Product role are less fault-prone than the classes that participate in the Concrete-Creator role and less fault-prone than the non-participant classes.

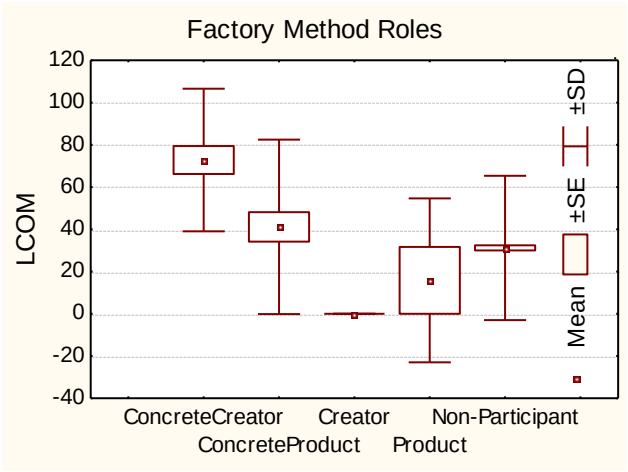
Table 5.11: Evaluation results of Factory Method pattern and its roles

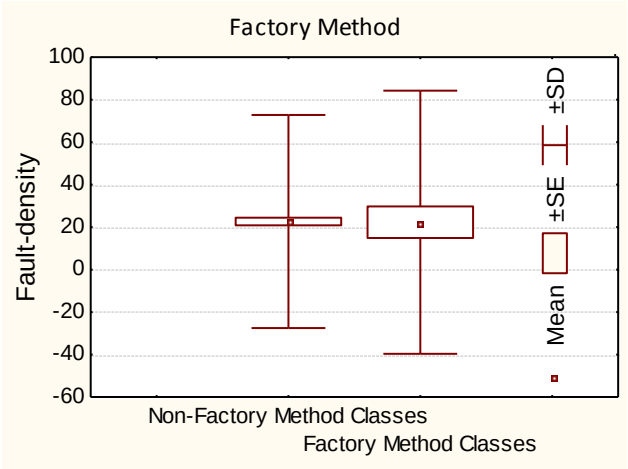
Factory Method	
CBO	
Factory Method Classes vs. Non-Factory Classes	**0.003325**
Overall Roles Comparison	**0.003**
Product vs. Concrete-Creator	**0.039**
Non-participant vs. Concrete-Creator	**0.019**
Non-participant vs. Concrete-Product	**0.002**
LCOM	
Factory Method Classes vs. Non-Factory Classes	**0.000389**
Overall Roles Comparison	**0.000**
Creator vs. Concrete-Creator	**0.001**
Product vs. Concrete-Creator	**0.000**
Non-participant vs. Concrete-Creator	**0.000**
Concrete-Product vs. Concrete-Creator	**0.001**
Fault-density	
Factory Method Classes vs. Non-Factory Classes	**0.037878**
Overall Roles Comparison	**0.034**
Concrete-Product vs. Non-participant	**0.003**
Fault-proneness	
Factory Method Classes vs. Non-Factory Classes	**0.028418**
Overall Roles Comparison	**0.011**
Concrete-Product vs. Non-participant	**0.001**
Concrete-Product vs. Concrete-Creator	**0.013**

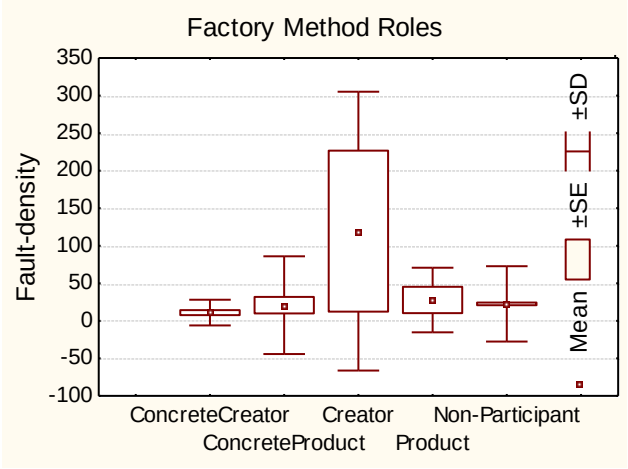

Factory Method
CBO
18
16
14
12
10
8
6
4
2
0
-2
-4
-6
-8
±SD
±SE
Mean
Non-Factory Method Classes
Factory Method Classes

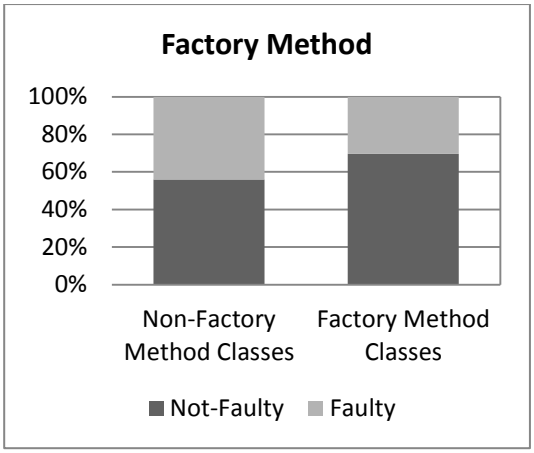

Factory Method Roles
CBO
60
50
40
30
20
10
0
-10
-20
-30
±SD
±SE
Mean
ConcreteCreator
ConcreteProduct
Creator
Product
Non-Participant


Factory Method
LCOM
100
80
60
40
20
0
-20
±SD
±SE
Mean
Non-Factory Method Classes
Factory Method Classes


Factory Method Roles
LCOM
120
100
80
60
40
20
0
-20
-40
±SD
±SE
Mean
ConcreteCreator
ConcreteProduct
Creator
Product
Non-Participant
Product


Factory Method
Fault-density
100
80
60
40
20
0
-20
-40
-60
±SD
±SE
Mean
Non-Factory Method Classes
Factory Method Classes


Factory Method Roles
Fault-density
350
300
250
200
150
100
50
0
-50
-100
±SD
±SE
Mean
ConcreteCreator
ConcreteProduct
Creator
Product
Non-Participant
Product



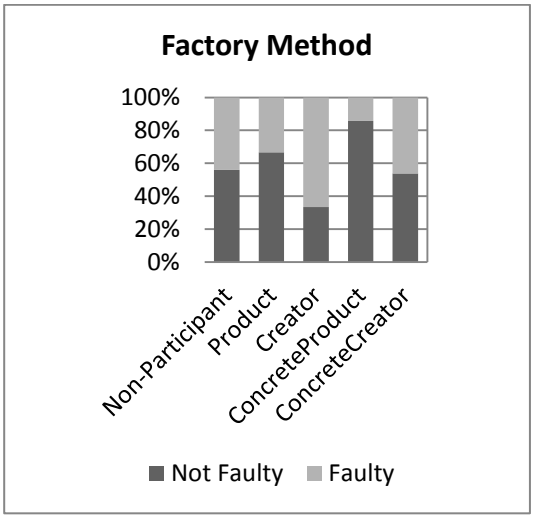


Figure 5.10: Comparison of coupling, cohesion, fault-density and fault-proneness of the Factory Method design pattern and its roles

5.1.3.1.3 Prototype

- **Coupling**

The p-value associated with evaluating the difference in coupling between the Prototype classes and the non-Prototype classes is significant as in table 5.12. This value indicates that there is significant difference in the coupling of the Prototype classes and the non-Prototype classes. As it can be seen in figure 5.11, the Prototype classes are more coupled than the non-Prototype classes.

As it can be seen in figure 5.11, there are some differences in the coupling among the different roles of the Prototype design pattern. However, the associated p-value is not significant as in table 5.12.

- **Cohesion**

The result of evaluating the difference in cohesion between the Prototype and the non-prototype classes results in a significant p-value as in table 5.12. This indicates that there is a significant difference in the cohesion of the Prototype and the non-prototype classes. As it can be seen in figure 5.11, the Prototype classes are less cohesive than the non-Prototype classes.

In evaluating the difference in cohesion among the classes that participate in the different roles of the Prototype design pattern, it was found that there is significant difference among these classes. The associated p-value with the evaluation the difference in cohesion among these roles is significant as in table 5.12. Three pairs of roles are associated with significant differences. These pairs are: Prototype vs. Client, Prototype vs. Concrete-Prototype and Non-participant vs. Concrete-Prototype. As it can be seen in figure 5.11, the classes that participate in the Prototype role are more cohesive than the classes that participate in the Client and Concrete-Prototype roles. Also, we can see that the non-participant classes are more cohesive than the classes that participate in the Concrete-Prototype role.

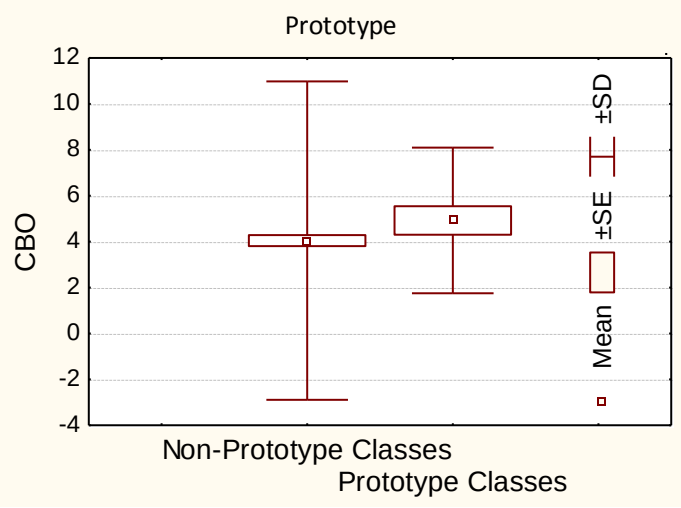
- **Fault-proneness and Fault-density**

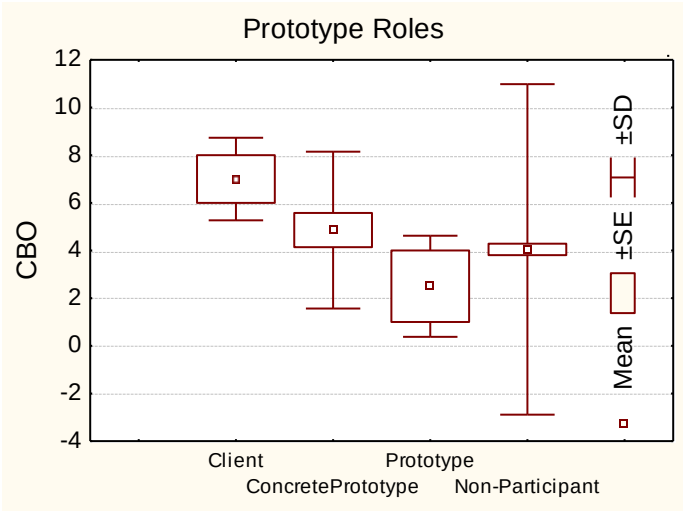
The p-values associated with evaluating fault-proneness and fault-density are insignificant as in table 5.12. As we can see, the obtained results indicate that there are no significant

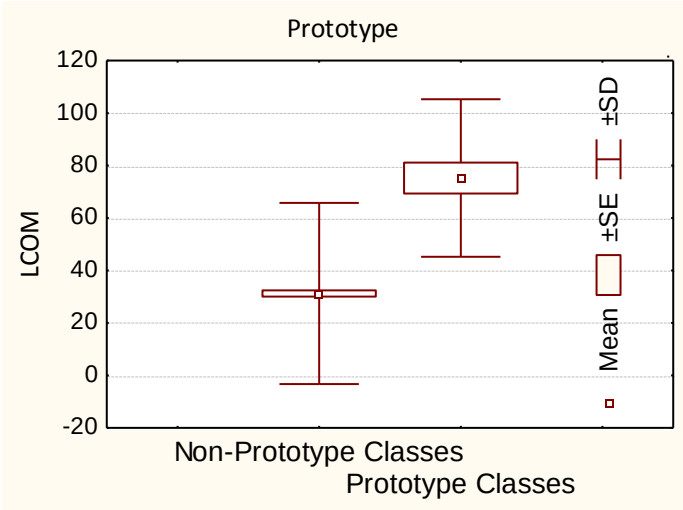
differences in the fault-proneness and fault-density on the pattern level and on the role level.

Table 5.12: Evaluation results of Prototype pattern and its roles

Prototype	
CBO	
Prototype classes vs. Non- Prototype Classes	**0.010384**
Overall Roles Comparison	0.57
LCOM	
Prototype classes vs. Non- Prototype Classes	**0.000000**
Overall Roles Comparison	**0.000**
Prototype - Client	**0.025**
Prototype vs. Concrete-Prototype	**0.002**
Non-participant vs. Concrete-Prototype	**0.000**
Fault-density	
Prototype classes vs. Non- Prototype Classes	0.221419
Overall Roles Comparison	0.665
Fault-proneness	
Prototype classes vs. Non- Prototype Classes	0.637303
Overall Roles Comparison	0.891







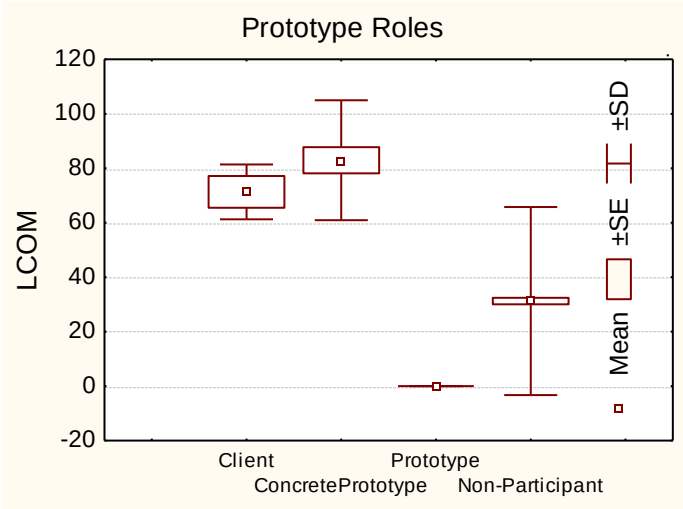


Figure 5.11: Comparison of coupling and cohesion of the Prototype design pattern and its roles

5.1.3.1.4 Singleton

We do not need to evaluate the different roles of the Singleton design pattern because it has only one role.

It can be seen in table 5.13 that the only significant value is associated with the evaluating the difference in coupling between the Singleton classes and the non-singleton classes. This indicates that there is a significant difference between the coupling of Singleton classes and the non-singleton classes. As it can be seen in figure 5.12, the singleton classes are more coupled than the non-singleton classes. In evaluating all the other attributes, we can see that the p-values associated with evaluating them are not significant.

Table 5.13: Evaluation results of Singleton pattern and its roles

Singleton	
CBO	
Singleton classes vs. Non- Singleton Classes	**0.013296**
LCOM	
Singleton classes vs. Non- Singleton Classes	0.067338
Fault-density	
Singleton classes vs. Non- Singleton Classes	0.773627
Fault-proneness	
Singleton classes vs. Non- Singleton Classes	0.995070

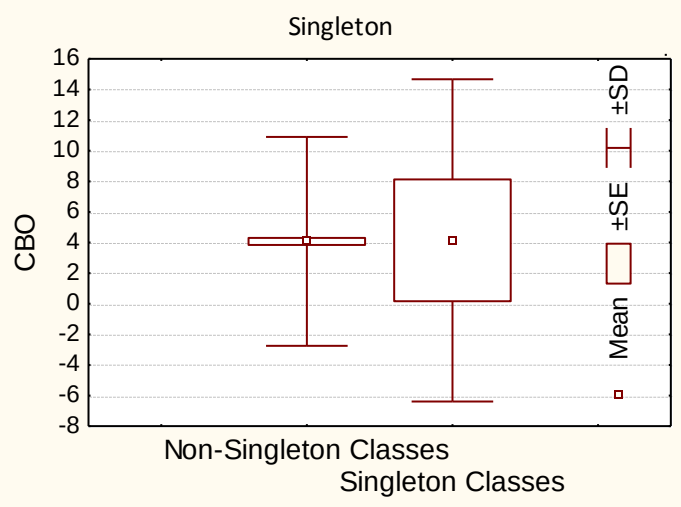


Figure 5.12: Comparison of coupling of the Singleton design pattern and its roles

5.1.3.2 Structural Design Patterns

5.1.3.2.1 Adapter

- **Coupling**

The distribution of coupling is not the same across the adapter classes and the non-adapter classes. It can be seen in table 5.14 that the p-value associated with evaluating the coupling of the adapter classes versus the non-adapter classes is significant. It suggests that there is a significant difference between the adapter classes and the non-adapter classes. As it can be seen in figure 5.13, the adapter classes are more coupled than the non-adapter classes.

The coupling distribution is not the same among the different roles of the adapter design pattern. It can be seen in table 5.14 that the p-value associated with the evaluating the overall difference among the classes that participate in the different roles in the adapter design pattern is significant.

Six different pairs of roles show significant differences as shown in table 5.14. These pairs are: Target vs. Non-participant, Target vs. Adapter, Target vs. Adaptee, Target vs. Client, Non-participant vs. Adaptee and Non-participant vs. Client. The evaluation of coupling of the other pairs results in insignificant p-values. As it can be seen in figure 5.13 that the classes that participate in the Target role are less coupled than the classes in the Adapter, Adaptee and Client roles and less coupled than the Non-participant classes. Also, we can see that the Non-participant classes are less coupled than the classes that participate in the Adaptee and Client roles.

- **Cohesion**

The obtained p-values associated with cohesion evaluation are not significant as it can be seen in table 5.14. Based on that, we can say that the cohesion distribution across the adapter and the non-adapter classes and across the different roles is almost the same. There is no significant difference between the adapter classes and the non-adapter classes and there is no significant difference among the classes that participate in the different roles of the adapter design pattern.

- **Fault-density**

The p-values associated with fault-density evaluation in the adapter design pattern are significant on the pattern level and on the role level as shown in table 5.14. As it can be seen in figure 5.13, the non-adapter classes are more fault-dense than the adapter classes.

In evaluating the fault-density of the classes that participate in the different roles of the adapter design pattern, we can see that two pairs of roles are associated with significant p-values. These pairs are: Adapter vs. Non-participant and Adapter vs. Client. As it can be

seen in figure 5.13, the classes that participate in the Adapter role are less fault-dense than the classes that participate in the Non-participant classes and the classes that participate in the Client role.

- **Fault-proneness**

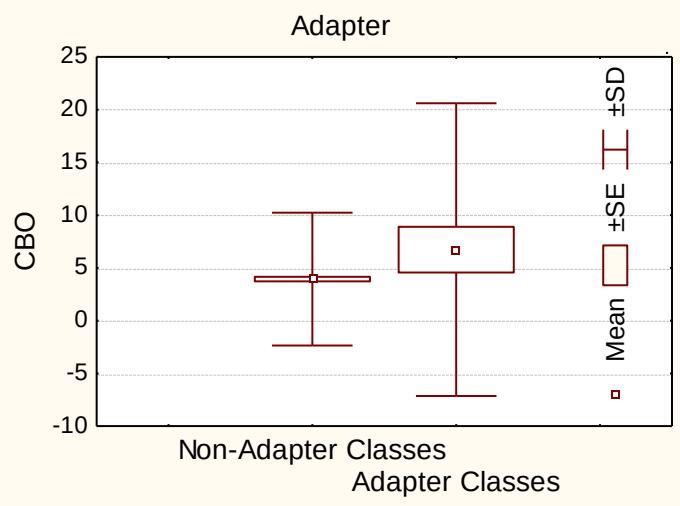
The p-values associated with fault-proneness evaluation reveal that there are significant differences in fault-proneness on the pattern level and on the role level as in table 5.14. As it can be seen in figure 5.13, the adapter classes are less fault-prone than the non-adapter classes.

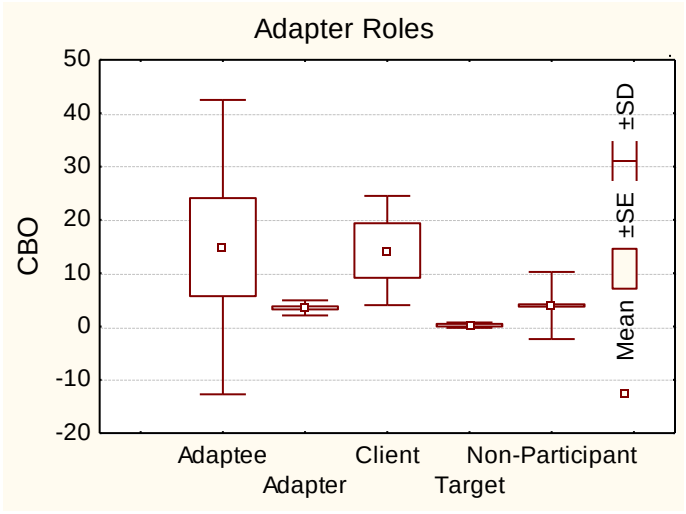
In evaluating the difference among the classes that participate in the different roles, it was found that there are three pairs that are associated with significant differences. These pairs are: Adapter vs. Non-participant, Adapter vs. Adaptee and Adapter vs. Client. As it can be seen in figure 5.13 that the classes that participate in the adapter design pattern are less fault-prone than the classes that participate in the Non-participant, Adaptee, and Client roles.

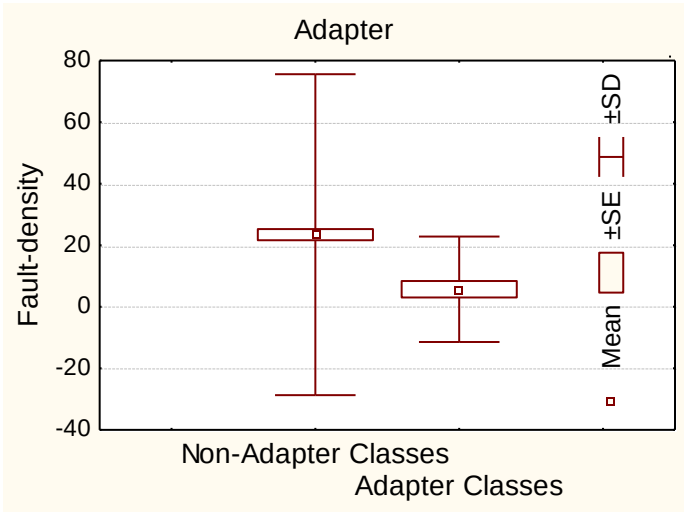
Table 5.14: Evaluation results of Adapter pattern and its roles

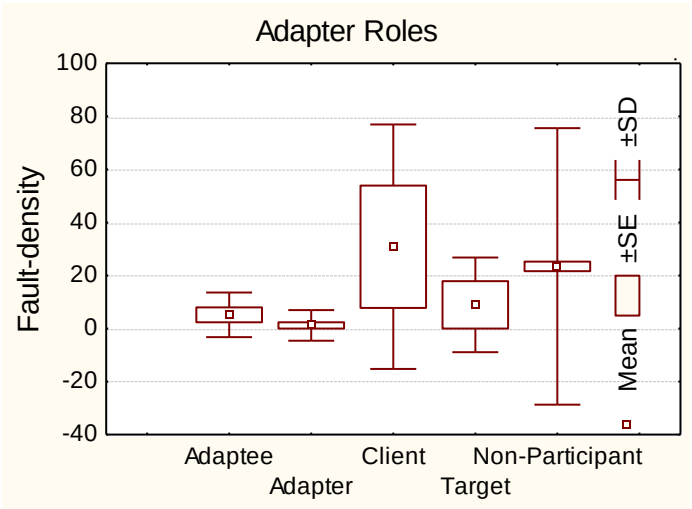
Adapter	
CBO	
Adapter classes vs. Non-Adapter classes	**0.030623**
Overall Roles Comparison	**0.000**
Target vs. Non-participant	**0.009**
Target vs. Adapter	**0.003**
Target vs. Adaptee	**0.000**
Target - Client	**0.000**
Non-participant vs. Adaptee	**0.015**
Non-participant vs. Client	**0.021**
LCOM	

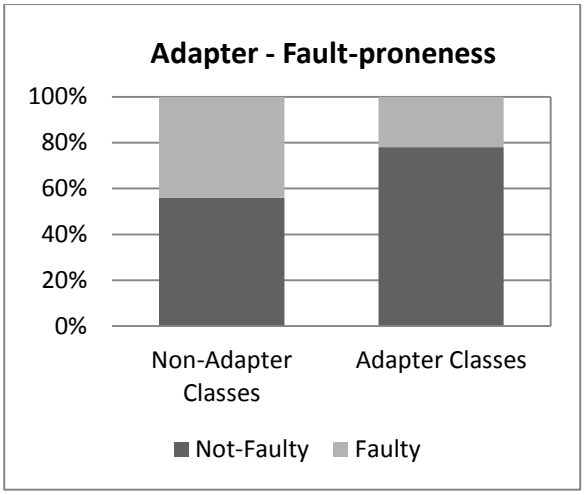
Adapter classes vs. Non-Adapter classes	0.350612
Overall Roles Comparison	0.3000
Fault-density	
Adapter classes vs. Non-Adapter classes	**0.001999**
Overall Roles Comparison	**0.003**
Adapter vs. Non-participant	**0.000**
Adapter - Client	**0.019**
Fault-proneness	
Adapter classes vs. Non-Adapter classes	**0.005370**
Overall Roles Comparison	**0.002**
Adapter vs. Non-participant	**0.000**
Adapter vs. Adaptee	**0.037**
Adapter vs. Client	**0.008**




Adapter Roles
CBO
50
40
30
20
10
0
-10
-20
±SD
±SE
Mean
Adaptee
Client
Non-Participant
Adapter
Target


Adapter
Fault-density
80
60
40
20
0
-20
-40
±SD
±SE
Mean
Non-Adapter Classes
Adapter Classes


Adapter Roles
Fault-density
100
80
60
40
20
0
-20
-40
±SD
±SE
Mean
Adaptee
Client
Non-Participant
Adapter
Target



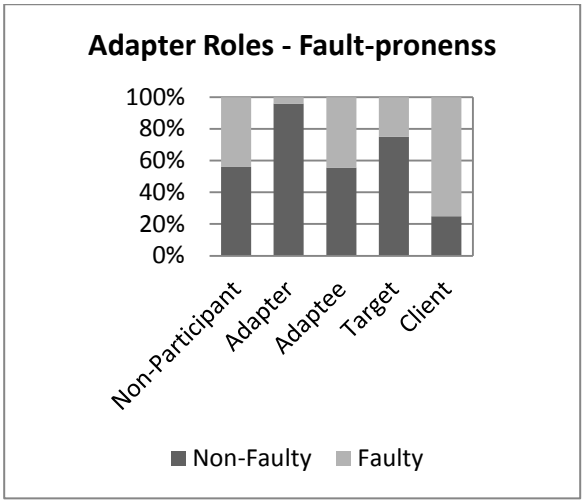


Figure 5.13: Comparison of coupling, fault-density and fault-proneness of the Adapter design pattern and its roles

5.1.3.2.2 Bridge

- **Coupling**

There is no significant difference in the distribution of coupling between the Bridge classes and the non-Bridge classes. It can be seen in table 5.15 that the p-value associated with evaluating the coupling of the classes that participate in the Bridge design pattern and the non-participant classes is not significant. However, the p-value associated with

evaluating the difference in coupling among the classes that participate in the different roles in the Bridge design pattern is significant as in table 5.15. Four different pairs of roles show significant differences. These pairs are: Implementor vs. Abstraction, Implementor vs. Refined-abstraction, Non-participant vs. Abstraction and Non-participant vs. Refined-abstraction.

We can see figure 5.14 that the Bridge classes are more coupled than the non-Bridge classes. Also, we can see that the classes that participate in the Abstraction and Refined Abstraction roles are more coupled than the classes that participate in the Implementor and the non-participant classes.

- **Cohesion**

The p-value associated with evaluating the difference in the cohesion of the classes that participate in the Bridge design pattern and the non-participant classes is significant as shown in table 5.15. As it can be seen in figure 5.14 that the classes that participate in the Bridge design pattern are less cohesive than the classes that participate in the non-Bridge classes.

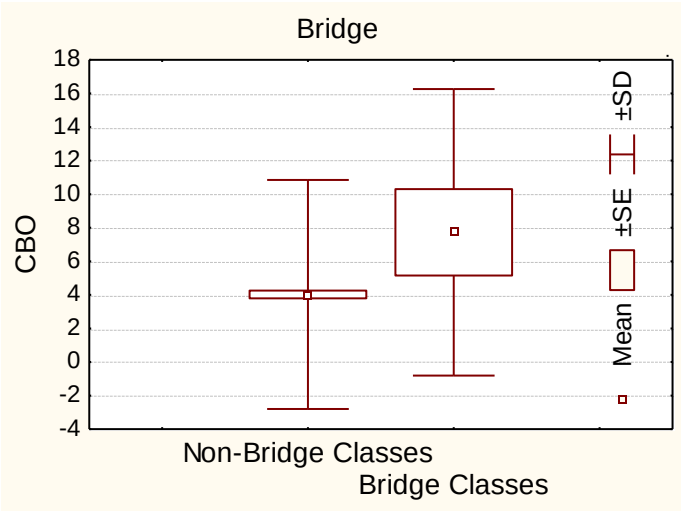
For the differences among roles, we can see that the p-value associated with the evaluating the cohesion among the different roles in the Bridge design pattern is significant as shown in table 5.15. Only one pair of roles is associated with a significant difference. This pair is Non-participant vs. Concrete-implementer. As it can be seen in figure 5.14, the Concrete Implementor classes are less cohesive than the non-participant classes.

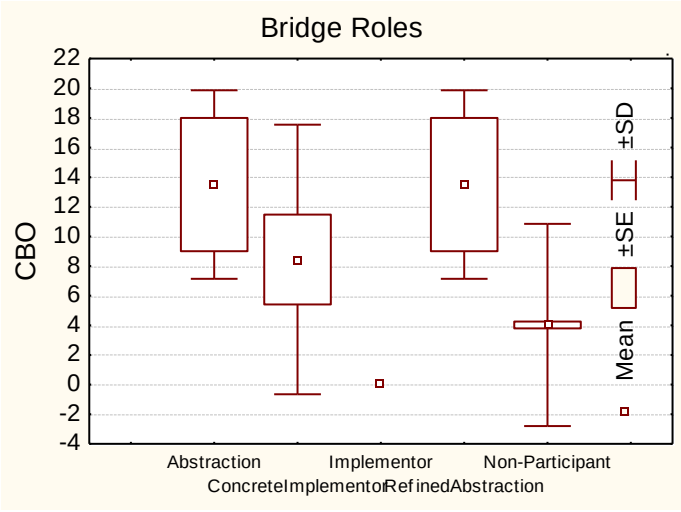
- **Fault-proneness and fault-density**

Evaluating the fault-proneness and the fault-density of the Bridge design pattern results in insignificant p-values as we can see in table 5.15. This indicates that there is no significant difference in evaluating fault-proneness and fault-density of the Bridge classes and the non-Bridge Classes. Also, it indicates that there is no significant difference in fault-proneness and fault-density among the roles of the Bridge classes.

Table 5.15: Evaluation results of Bridge pattern and its roles

Bridge	
CBO	
Bridge classes vs. Non-Bridge classes	0.120628
Overall Roles Comparison	**0.005**
Implementor vs. Abstraction	**0.014**
Implementor vs. Refined-abstraction	**0.014**
Non-participant vs. Abstraction	**0.026**
Non-participant vs. Refined-abstraction	**0.026**
LCOM	
Bridge classes vs. Non-Bridge classes	**0.014623**
Overall Roles Comparison	**0.008**
Non-participant vs. Concrete-implementer	**0.006**
Fault-density	
Bridge classes vs. Non-Bridge classes	0.400068
Overall Roles Comparison	0.758
Fault-proneness	
Bridge classes vs. Non-Bridge classes	0.867159
Overall Roles Comparison	0.196


Bridge
CBO
18
16
14
12
10
8
6
4
2
0
-2
-4
±SD
±SE
Mean
Non-Bridge Classes
Bridge Classes


Bridge Roles
CBO
22
20
18
16
14
12
10
8
6
4
2
0
-2
-4
±SD
±SE
Mean
Abstraction
Implementor
Non-Participant
ConcreteImplementor
RefinedAbstraction


Bridge
LCOM
100
80
60
40
20
0
-20
±SD
±SE
Mean
Non-Bridge Classes
Bridge Classes

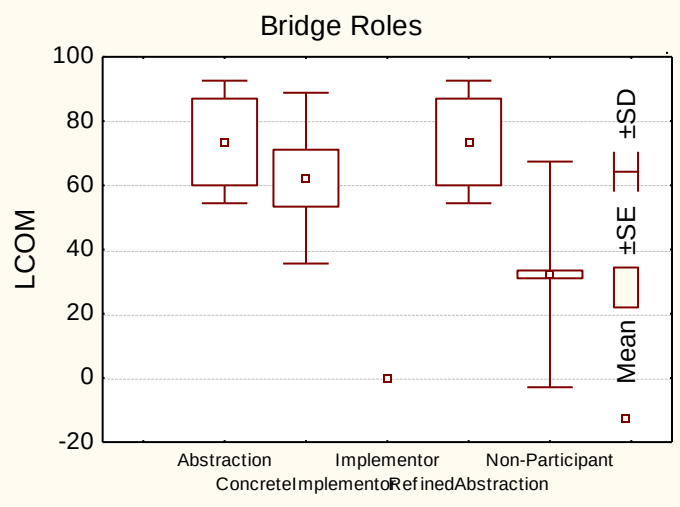


Figure 5.14: Comparison of coupling and cohesion of the Bridge design pattern and its roles

5.1.3.2.3 Composite

- **Coupling**

There is no significant difference in evaluating the coupling of the classes that participate in the Composite design pattern and the non-participant classes. The p-value associated with evaluating the difference in coupling between the classes that participate in the Composite design pattern and the non-participant classes is insignificant as it can be seen in table 5.16. However, the p-value associated with evaluating the difference among the different roles of the composite design pattern is significant as in table 5.16. Four different pairs of roles show significant differences. These pairs are: Component vs. Client, Leaf vs. Client, Non-participant vs. Client and Composite vs. Client. As it can be seen in figure 5.15, the classes that participate in the Client role are more coupled than the classes that participate in the Component, Composite and Leaf roles and more coupled than the non-composite classes as well.

- **Cohesion**

The p-value associated with evaluating the difference in the cohesion of the Composite classes and the non-Composite classes is significant as shown in table 5.16. This indicates that there is a significant difference in the cohesion of the classes that participate in the Composite design pattern and the classes that do not participate. As it can be seen in figure 5.15, the non-Composite classes are more cohesive than the Composite classes.

The p-value associated with evaluating the difference in the cohesion among the classes that participate in the different roles is significant as it can be seen in table 5.16. We can see that there are six different pairs of roles associated with significant differences. These roles are: Component vs. Leaf, Component vs. Composite, Component vs. Client, Non-participant vs. Leaf, Non-participant vs. Composite, and Non-participant vs. Client. As it can be seen in figure 5.15, the classes that participate in the Component role and the non-participant classes are more cohesive than the classes that participate in the Leaf, Composite and Client roles.

- **Fault-density**

The p-value associated with evaluating the difference in fault-density of the Composite classes and the non-Composite classes is significant as it can be seen in table 5.16. This value indicates that there is a significant difference in fault-density between the classes that participate in the Composite design pattern and the non-participant classes. As it can be seen in figure 5.15, the Composite classes are less fault-dense than the non-Composite classes.

Regarding the evaluation of the fault-density of the different roles of the Composite design pattern, the associated p-value is significant as in table 5.16. As it can be seen in table 5.16, there is only one pair of roles that show significant difference. This pair is: Leaf vs. Non-participant. As it can be seen in figure 5.15, the non-participant classes are more fault-dense than the classes that participate in the Leaf role.

- **Fault-proneness**

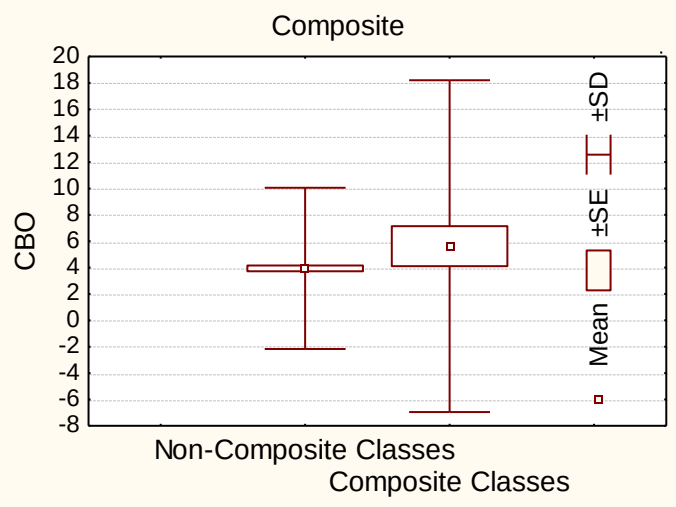
The p-value associated with evaluating the difference in fault-proneness of the Composite classes and the non-Composite classes is significant as shown in table 5.16. It indicates that there is a significant difference in fault-proneness of the classes that participate in the Composite design pattern and the non-participant classes. As it can be seen in figure 5.15, the classes that participate in the Composite design pattern are less fault-prone than the non-Composite classes.

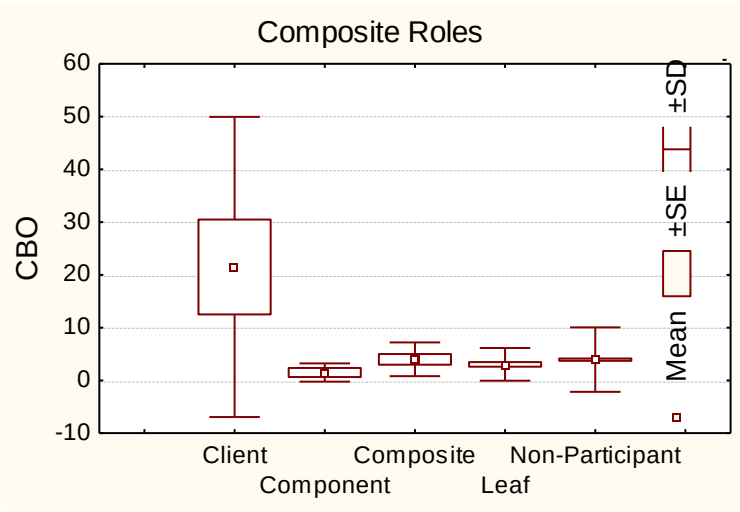
In evaluating the difference in fault-proneness among the different roles of the Composite design pattern, it was found that there is a significant difference in the fault-proneness among the different roles. This is clear from the associated p-value as shown in the table 5.16. Only one pair of roles shows significant difference. This role is: Leaf vs. Non-participant. We can see from figure 5.15 that the classes that participate in the Leaf role are less fault-prone than the non-participant classes.

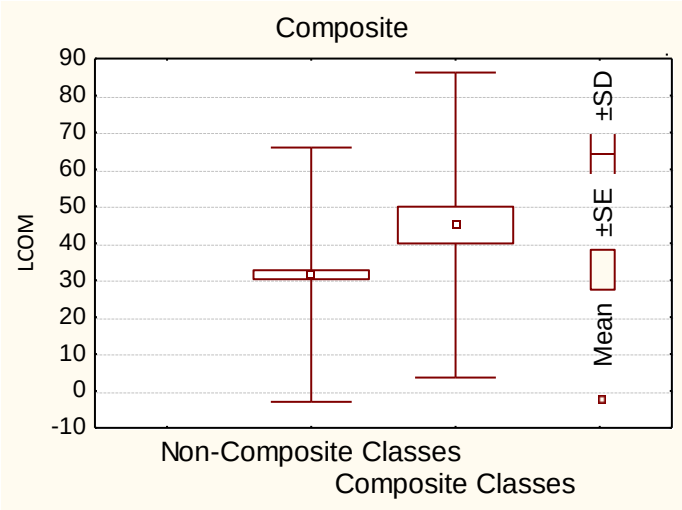
Table 5.16: Evaluation results of Composite pattern and its roles

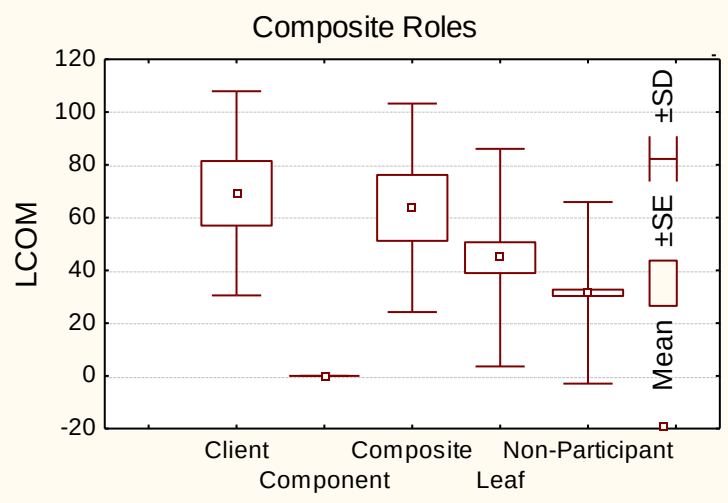
Composite	
CBO	
Composite classes vs. Non-Composite Classes	0.841107
Overall Roles Comparison	**0.003**
Component - Client	**0.003**

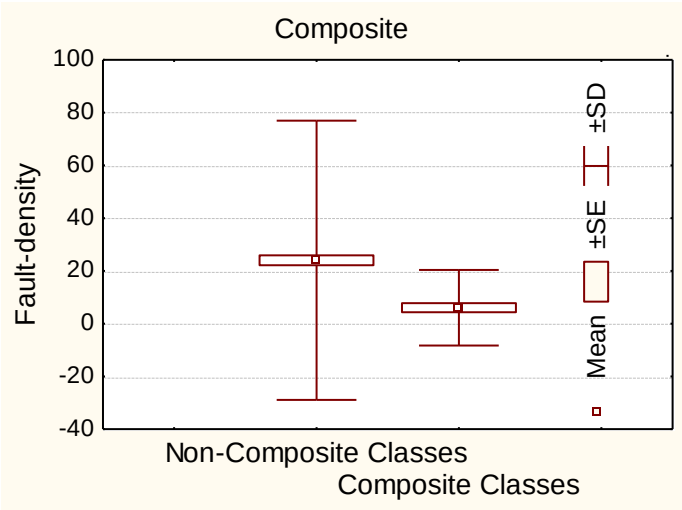
Leaf vs. Client	**0.002**
Non-participant vs. Client	**0.001**
Composite vs. Client	**0.044**
LCOM	
Composite classes vs. Non-Composite Classes	**0.003714**
Overall Roles Comparison	**0.000**
Component - Leaf	**0.014**
Component - Composite	**0.002**
Component - Client	**0.001**
Non-participant vs. Leaf	**0.013**
Non-participant vs. Composite	**0.005**
Non-participant vs. Client	**0.001**
Fault-density	
Composite classes vs. Non-Composite Classes	**0.000418**
Overall Roles Comparison	**0.001**
Leaf vs. Non-participant	**0.000**
Fault-proneness	
Composite classes vs. Non-Composite Classes	**0.003163**
Overall Roles Comparison	**0.009**
Leaf vs. Non-participant	**0.003**

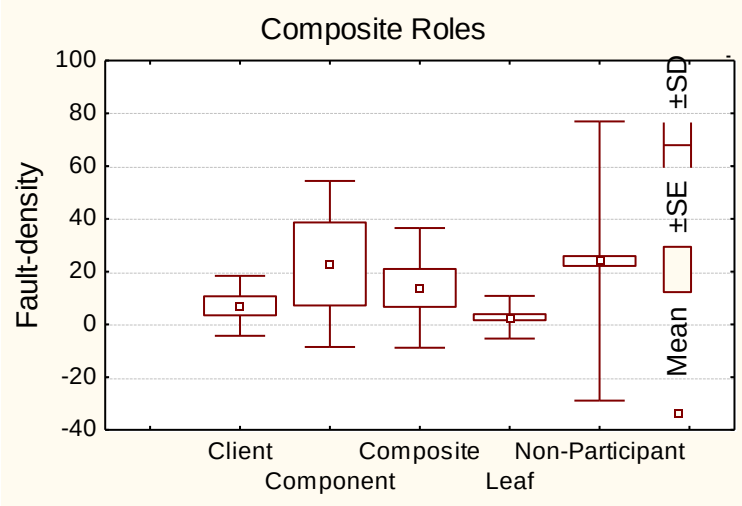


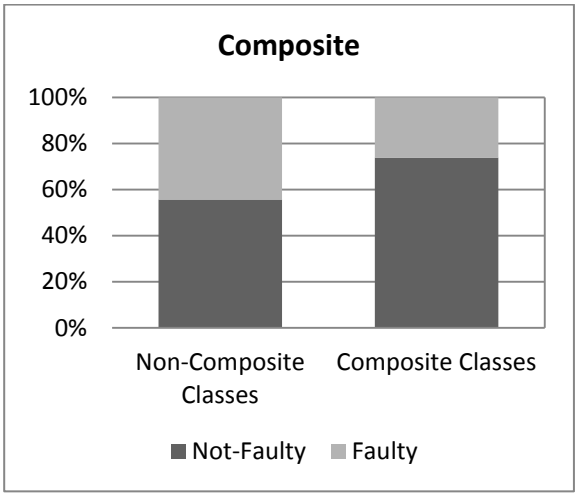

Composite Roles
CBO
60
50
40
30
20
10
0
-10
Client
Composite
Non-Participant
Component
Leaf
±SD
±SE
Mean


Composite
LCOM
90
80
70
60
50
40
30
20
10
0
-10
Non-Composite Classes
Composite Classes
±SD
±SE
Mean


Composite Roles
LCOM
120
100
80
60
40
20
0
-20
Client
Composite
Non-Participant
Component
Leaf
±SD
±SE
Mean


Composite
100
80
60
40
20
0
-20
-40
Fault-density
±SD
±SE
Mean
Non-Composite Classes
Composite Classes


Composite Roles
100
80
60
40
20
0
-20
-40
Fault-density
±SD
±SE
Mean
Client Component
Composite Leaf
Non-Participant


Composite
100%
80%
60%
40%
20%
0%
Non-Composite Classes
Composite Classes
Not-Faulty
Faulty

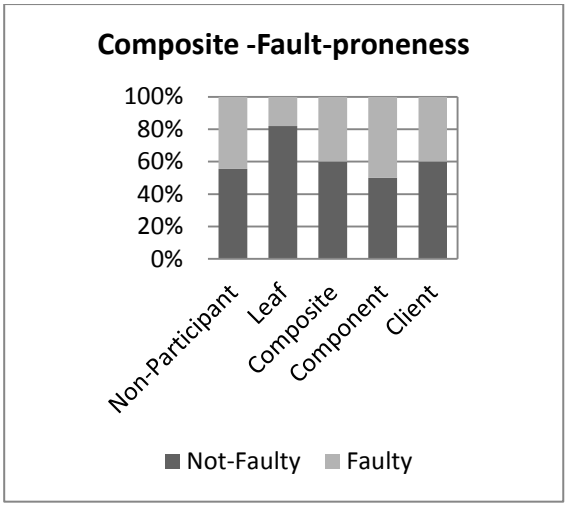


Figure 5.15: Comparison of coupling, cohesion, fault-proneness and fault-density of the Adapter design pattern and its roles

5.1.3.2.4 Decorator

- **Coupling and cohesion**

As it can be seen in table 5.17, the p-values associated with evaluating the differences in coupling and cohesion in the Decorator design pattern are not significant. The results suggest that there are no significant differences in coupling and cohesion between the Decorator classes and the non-Decorator classes. The same thing can be said about the differences in the coupling and cohesion among the different roles of the Decorator design pattern - there are no significant differences among the different roles of the Decorator design pattern.

- **Fault-density**

The evaluation of fault-density of Decorator classes versus the non-Decorator classes results in a significant p-value. This indicates that there is a significant difference in fault-density between the Decorator classes and the non-Decorator classes. As it can be seen in figure 5.16, the Decorator classes are less fault-dense than the non-Decorator classes.

In evaluating the fault-density of the classes that participate in the different roles of Decorator Design pattern, it was found that there is a significant difference among the different roles as the associated p-values indicate in table 5.17. Two different pairs show significant differences. These pairs are: Concrete-Decorator vs. Non-Participant and Concrete-Component vs. Non-participant. As it can be seen in figure 5.16, the classes that participate in Concrete-Component and Concrete-Decorator are less fault-dense than the non-participant classes.

- **Fault-proneness**

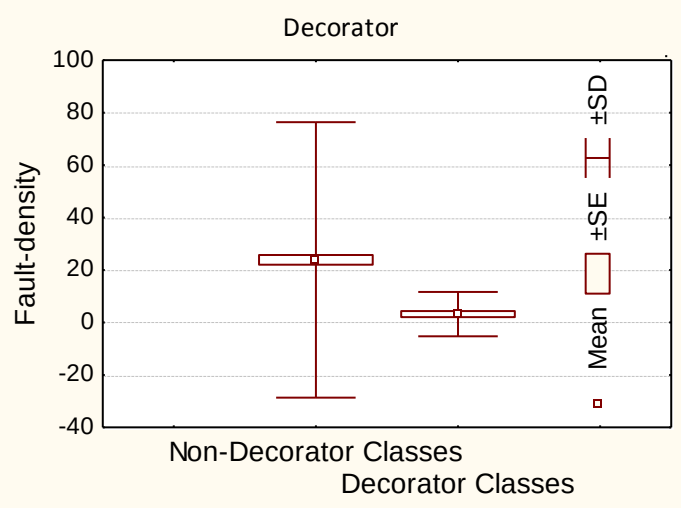
The p-value associated with evaluating the difference in fault-proneness between the Decorator classes and the non-Decorator classes is significant as it can be seen in table 5.17. The obtained p-value suggests that there is a significant difference between the Decorator and non-Decorator classes. As it can be seen in figure 5.16, the Decorator classes are less fault-prone than the non-Decorator classes.

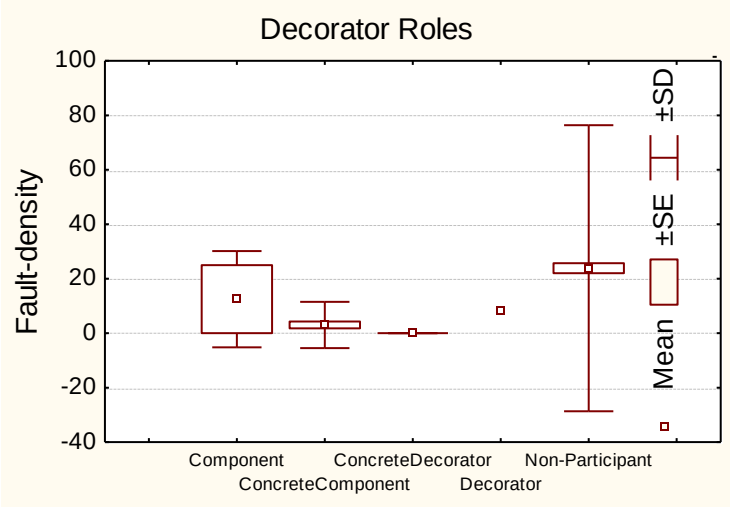
The evaluation of fault-proneness among the different roles results in a significant difference. This is clear from the obtained p-value in table 5.17. Two pairs of roles show significant differences as it can be seen in table 5.17. These roles are Concrete-Decorator vs. Non-Participant and Concrete-Component vs. Non-participant. As it can be seen in figure 5.16, the classes that participate in the Concrete-Decorator and Concrete-Component are less fault-prone than the non-participant classes.

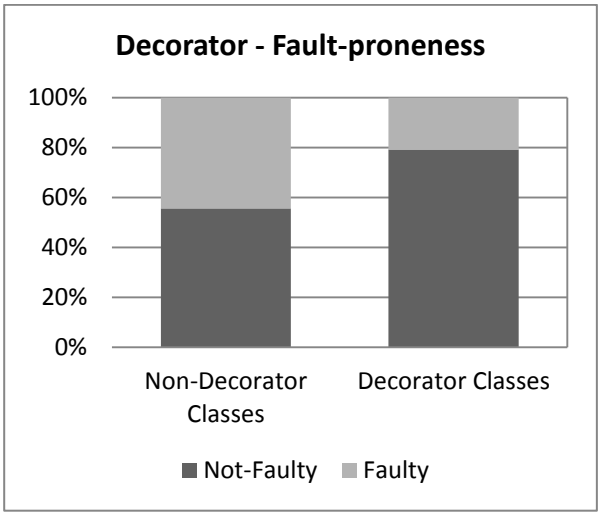
Table 5.17: Evaluation results of Decorator pattern and its roles

Decorator	
CBO	
Decorator classes vs. Non-Decorator Classes	0.249034

Overall Roles Comparison	0.541
LCOM	
Decorator classes vs. Non-Decorator Classes	0.053443
Overall Roles Comparison	0.130
Fault-density	
Decorator classes vs. Non-Decorator Classes	**0.000083**
Overall Roles Comparison	**0.001**
Concrete-Decorator vs. Non-Participant	**0.036**
Concrete-Component vs. Non-participant	**0.000**
Fault-proneness	
Decorator classes vs. Non-Decorator Classes	**0.000752**
Overall Roles Comparison	**0.003**
Concrete-Decorator vs. Non-Participant	**0.029**
Concrete-Component vs. Non-participant	**0.001**







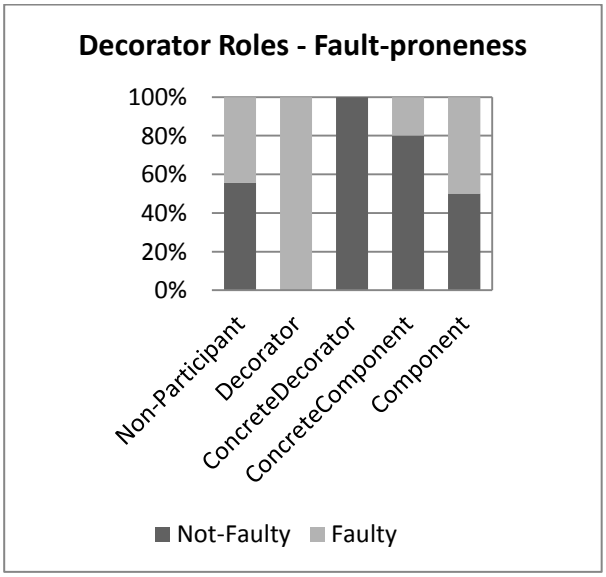


Figure 5.16: Comparison of fault-proneness and fault-density of the Decorator design pattern and its roles

5.1.3.2.5 Proxy

There are no significant differences in coupling, cohesion, fault-proneness and fault-density on the pattern level and on the role level. This is clear from the obtained p-values in table 5.18. These values indicate that there are no significant differences between the Proxy and the non-Proxy classes. Also, it indicates that there are no significant differences among the different roles of the Proxy design pattern.

Table 5.18: Evaluation results of Proxy pattern and its roles

Proxy	
CBO	
Proxy classes vs. Non- Proxy Classes	0.796128
Overall Roles Comparison	0.157
LCOM	
Proxy classes vs. Non- Proxy Classes	0.171788
Overall Roles Comparison	0.93
Fault-density	
Proxy classes vs. Non- Proxy Classes	0.152696
Overall Roles Comparison	0.563
Fault-proneness	
Proxy classes vs. Non- Proxy Classes	0.132256
Overall Roles Comparison	0.519

5.1.3.3 Behavioral Design Patterns

5.1.3.3.1 Command

- **Coupling**

There are significant differences between the coupling of the Command and the non-Command classes and among the classes that participate in the different roles of Command design pattern. It can be seen in table 5.19 that the p-value associated with

evaluating the difference in the coupling of the Command and the non-Command classes is significant. As it can be seen in figure 5.17, the classes that participate in the Command design pattern are more coupled than the non-participant classes.

The same thing can be said about the p-value associated with evaluating the coupling of the classes that participate in the different roles of the Command Design pattern as it can be seen in table 5.19. Three pairs of roles show significant differences. These pairs are: Non-participant vs. Client, Concrete-command vs. Client and Invoker vs. Client. As it can be seen in figure 5.17, the classes that participate in the Client role are more coupled than the classes that participate in the Concrete-Command and Invoker roles and more coupled than the non-participant classes.

- **Cohesion**

There is no significant difference between the cohesion of the classes that participate in the Command design pattern and the non-Command classes. It can be seen in table 5.19 that the associated p-value is not significant. However, the comparison of the different roles in the Command design pattern results in significant differences. It can be seen in table 5.19 that six pairs of roles show significant differences. These pairs are: Concrete-Command vs. Invoker, Concrete-Command vs. Command, Concrete-Command vs. Client, Receiver vs. Client, Non-participant vs. Command and Non-participant vs. Client. As it can be seen in figure 5.17 that the Classes that participate in the Concrete Command roles are more cohesive than the classes that participate in the Client, Command and Invoker roles. Also, we can see in that the classes that participate in the Receiver role are more cohesive than the classes that participate in the Client role. Moreover, we can see

that the non-participant classes in the Command design pattern are more cohesive than the classes that participate in the Client and Command Roles.

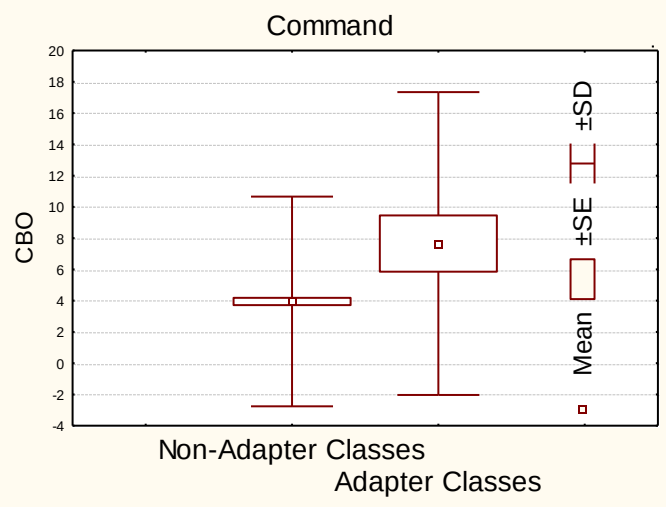
- **Fault-proneness and fault-density**

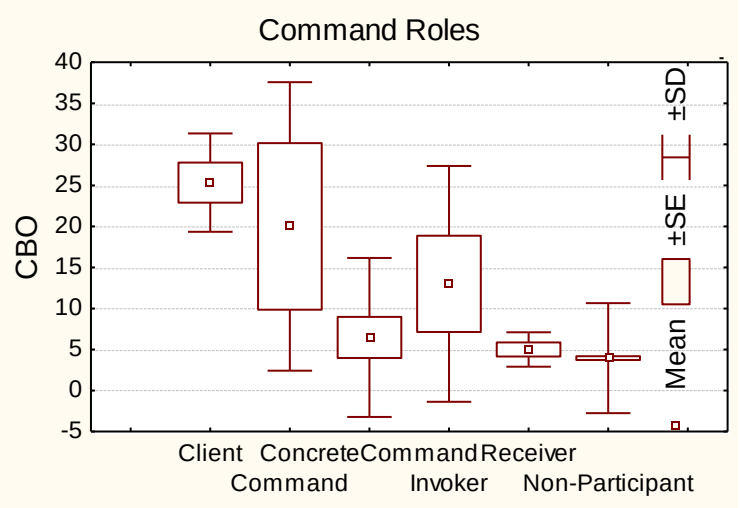
The p-values associated with evaluating the fault-proneness and density of the Command design pattern are insignificant on the pattern level and on the role level as shown in table 5.19. These values suggest that there is no significant difference in the fault-proneness and density of the classes that participate in the Command pattern and the non-participant classes. Also, they suggest that there are no significant differences in the fault-proneness and density among the different roles of the Command design pattern.

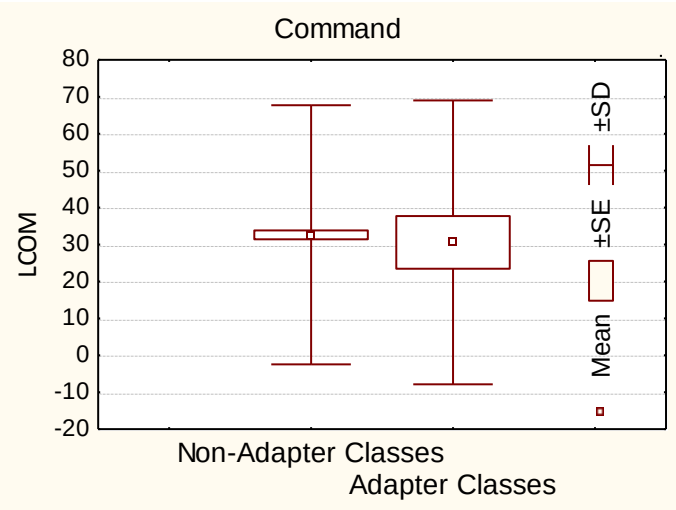
Table 5.19: Evaluation results of Command pattern and its roles

Command	
CBO	
Command classes vs. non-Command classes	**0.035873**
Overall Roles Comparison	**0.000**
Non-participant vs. Client	**0.000**
Concrete-command - Client	**0.003**
Invoker - Client	**0.023**
LCOM	
Command classes vs. non-Command classes	0.732659
Overall Roles Comparison	**0.000**
Concrete-Command - Invoker	**0.049**
Concrete-Command vs. Command	**0.011**
Concrete-Command vs. Client	**0.000**
Receiver vs. Client	**0.004**
Non-participant - Command	**0.045**
Non-participant - Client	**0.000**
Fault-density	
Command classes vs. non-Command classes	0.338768
Overall Roles Comparison	0.671

Fault-proneness	
Command classes vs. non-Command classes	0.577196
Overall Roles Comparison	0.092







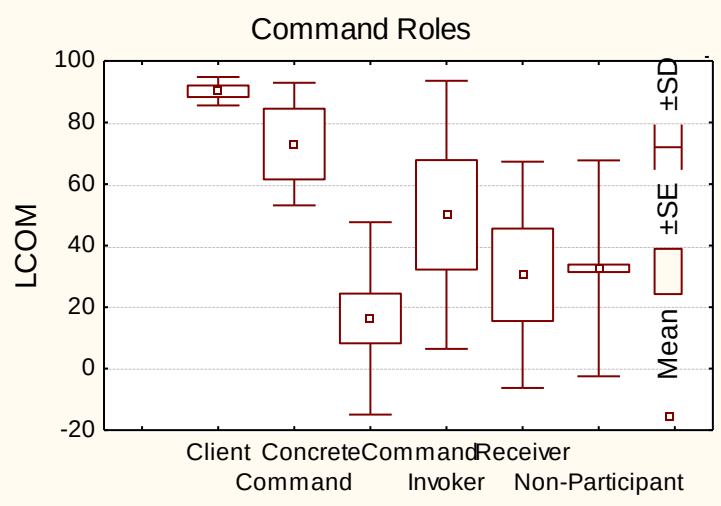


Figure 5.17: Comparison of coupling and cohesion of the Command design pattern and its roles

5.1.3.3.2 Iterator

- **Cohesion**

The p-value associated with evaluating the difference in the cohesion between the non-iterator classes and the iterator classes is significant as it can be seen in table 5.20. This indicates that there is a significant difference in cohesion between the Iterator and the

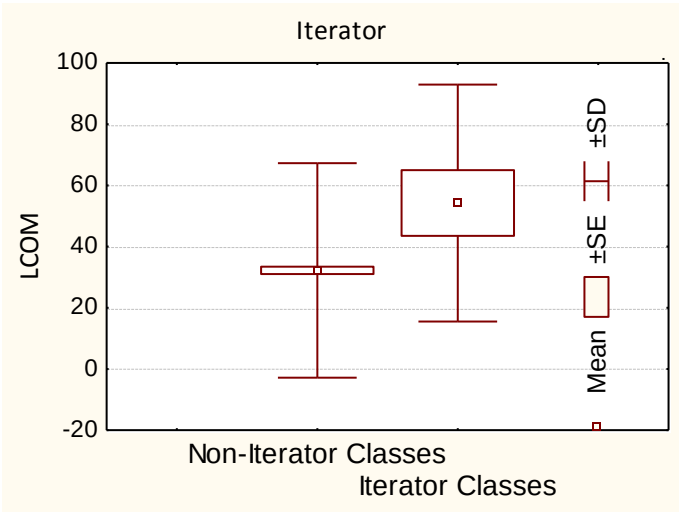
non-Iterator classes. As it can be seen in figure 5.18, the Iterator classes are less cohesive than the non-Iterator classes.

The evaluation of the difference among the classes that participate in the different roles of the iterator design pattern results in a significant p-value as it can be seen in table 5.20. The obtained p-value suggests that there is significant difference in the cohesion of the classes that participate in the different roles of the Iterator design pattern. There are five pairs of roles that show significant differences. These roles are: Aggregate vs. Concrete-Aggregator, Non-participant vs. Concrete-Aggregator, Iterator vs. Concrete-Iterator, Aggregate vs. Concrete-Iterator and Non-participant vs. Concrete-Iterator. As it can be seen in figure 5.18, the classes that participate in the Concrete-Aggregator role are less cohesive than the classes that participate in the Aggregate role and the non-participant classes. Also, we can see that the classes that participate in the Concrete-Iterator role are less cohesive than the classes that participate in the Iterator and Aggregate roles and less cohesive than the non-participant classes.

Table 5.20: Evaluation results of Iterator pattern and its roles

Iterator	
CBO	
Iterator classes vs. Non- Iterator Classes	0.063420
Overall Roles Comparison	0.100
LCOM	
Iterator classes vs. Non- Iterator Classes	**0.030735**
Overall Roles Comparison	**0.013**
Aggregate vs. Concrete-Aggregator	**0.024**
Non-participant vs. Concrete-Aggregator	**0.012**
Iterator vs. Concrete-Iterator	**0.046**
Aggregate vs. Concrete-Iterator	**0.035**
Non-participant vs. Concrete-Iterator	**0.019**

Fault-density	
Iterator classes vs. Non- Iterator Classes	0.785375
Overall Roles Comparison	0.549
Fault-proneness	
Iterator classes vs. Non- Iterator Classes	0.815496
Overall Roles Comparison	0.612



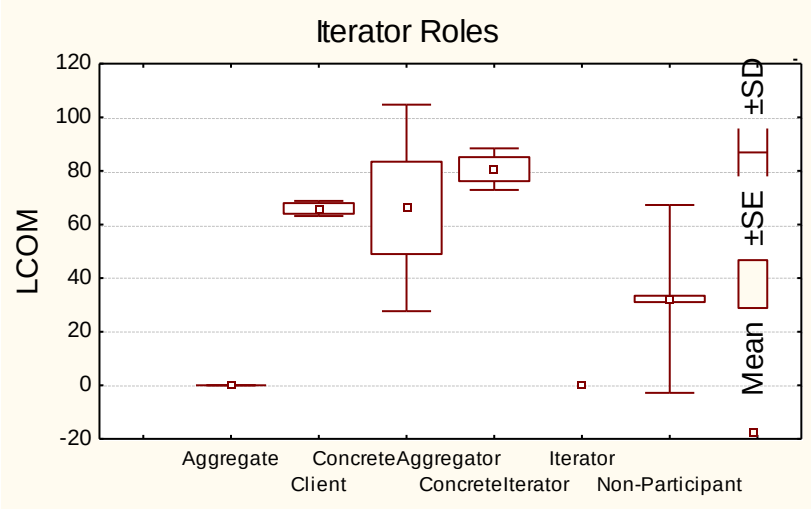


Figure 5.18: Comparison of cohesion of the Iterator design pattern and its roles

5.1.3.3.3 Memento

- **Coupling**

The evaluation of the difference in coupling between the Memento classes and the non-Memento classes results in a significant p-value as it can be seen in table 5.21. This indicates that there is a significant difference in coupling of the classes that participate in the Memento design pattern and the non-participant classes. As it can be seen in figure 5.19, the Memento classes are more coupled than the non-Memento classes.

In evaluating the difference among the different roles in the Memento design pattern, it can be seen in table 5.21 that the associated p-value is significant. This indicates that there is a significant difference in the coupling among the classes that participate in the different roles of the Memento design pattern. Two pairs of roles show significant differences. These pairs are: Non-participant vs. Caretaker and Non-participant vs. Originator. As it can be seen in figure 5.19 that the classes that participate in the Caretaker and in the Originator roles are more coupled than the non-participant classes.

- **Cohesion**

The evaluation of the difference in the cohesion between the Memento and the non-Memento classes results in a significant p-value as it can be seen in table 5.21. This value indicates that there is a significant difference in the cohesion of the Memento classes and the non-Memento classes. As it can be seen in figure 5.19, the Memento classes are less cohesive than the non-Memento classes.

Also, the p-value associated with evaluating the difference in the cohesion among the classes that participate in the different roles of the Memento classes is significant as it can

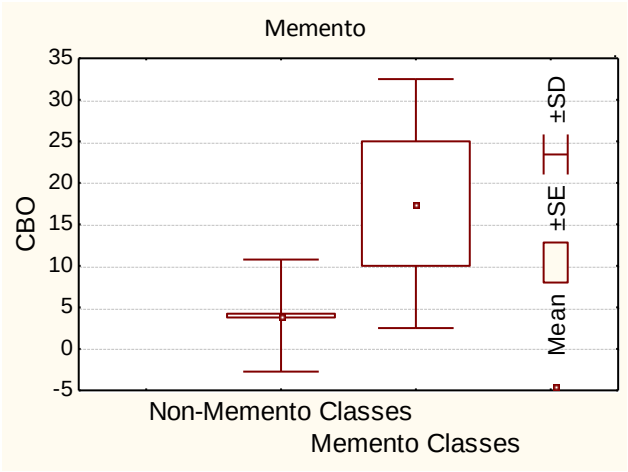
be seen in table 5.21. This value suggests that there is a significant difference in the cohesion of the different roles of the Memento design pattern. Only one pair is associated with significant difference. This pair is: Non-participant vs. Originator. As it can be seen in figure 5.19, the classes that participate in the Originator role are less cohesive than the non-participants classes.

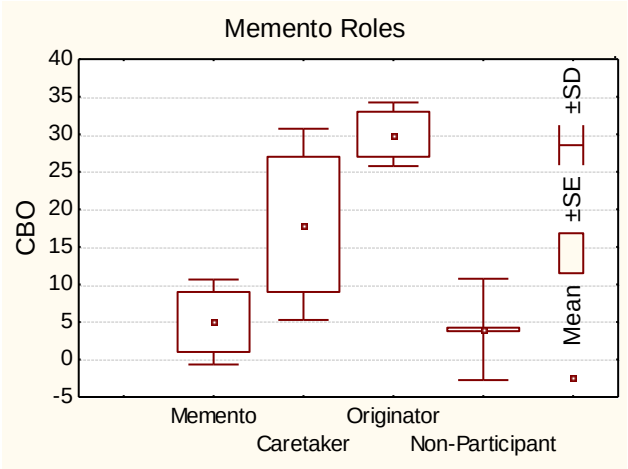
- **Fault-proneness and fault-density**

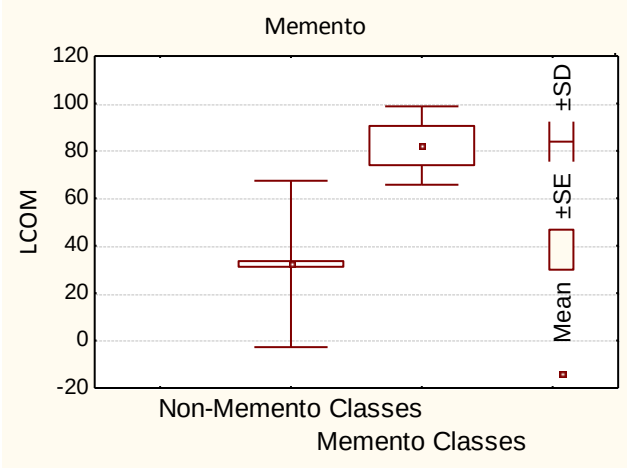
The evaluation of the fault-proneness density results in insignificant differences on both levels - the pattern level and role level. We can see in the table 5.21 that the p-values associated with the evaluating the differences in fault-proneness and density are insignificant.

Table 5.21: Evaluation results of Memento pattern and its roles

Memento	
CBO	
Memento classes vs. Non- Memento Classes	**0.046738**
Overall Roles Comparison	**0.011**
Non-participant vs. Caretaker	**0.025**
Non-participant vs. Originator	**0.015**
LCOM	
Memento classes vs. Non- Memento Classes	**0.005625**
Overall Roles Comparison	**0.017**
Non-participant vs. Originator	**0.035**
Fault-density	
Memento classes vs. Non- Memento Classes	0.829893
Overall Roles Comparison	0.829
Fault-proneness	
Memento classes vs. Non- Memento Classes	0.194909
Overall Roles Comparison	0.149


Memento
CBO
35
30
25
20
15
10
5
0
-5
±SD
±SE
Mean
Non-Memento Classes
Memento Classes


Memento Roles
CBO
40
35
30
25
20
15
10
5
0
-5
±SD
±SE
Mean
Memento
Caretaker
Originator
Non-Participant


Memento
LCOM
120
100
80
60
40
20
0
-20
±SD
±SE
Mean
Non-Memento Classes
Memento Classes

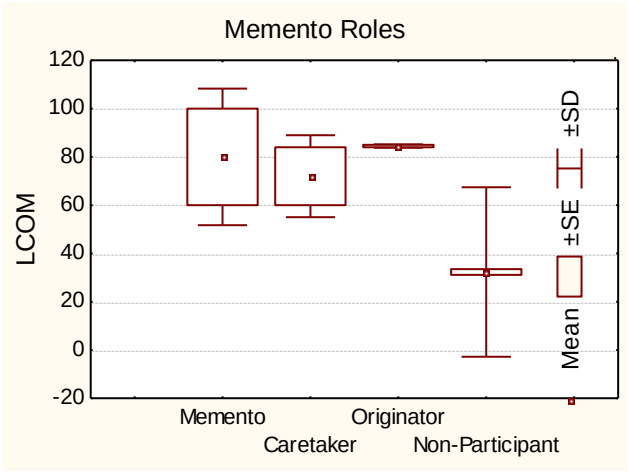


Figure 5.19: Comparison of coupling and cohesion of the Memento design pattern and its roles

5.1.3.3.4 Observer

- **Coupling**

The p-value associated with evaluating the difference in coupling between the Observer and the non-Observer classes is significant as it can be seen in table 5.22. This value suggests that there is a significant difference between the classes that participate in the Observer pattern and the non-Observer classes. As it can be seen in figure 5.20, the classes that participate in the Observer design pattern are more coupled than the non-participant classes.

Evaluating the difference in coupling among the different roles results in significant differences among the different roles as the associated p-value indicates. It can be seen in table 5.22 that there are five pairs of roles associated with significant p-values. These pairs are: Observer vs. Concrete-Observer, Observer vs. Concrete-Subject, Observer vs. Subject, Non-participant vs. Concrete-Observer and Non-participant vs. Concrete-Subject. As it can be seen in figure 5.20, the classes that participate in the Observer role are less

coupled than the classes that participate in the Concrete-Observer, Concrete-Subject and Subject roles. Also, we can see that the non-participant classes are less coupled than the classes that participate in the Concrete-Observer and Concrete-Subject roles.

- **Cohesion**

There is a significant difference in the cohesion of the Observer classes and the non-Observer classes as the associated p-value indicates. It can be seen in table 5.22 that the p-value associated with evaluating the difference in the cohesion between the Observer and the non-Observer classes is significant. As it can be seen in figure 5.20, the classes that participate in the Observer design pattern are less cohesive than the non-participant classes.

The obtained results in evaluating the difference among the different roles of the Observer design pattern result in a significant p-value as we can see in the table 5.22. We can see that there is a significant difference in the cohesion among the different roles of the Observer design pattern. Six pairs of roles show significant differences. These pairs are: Observer vs. Non-participant، Observer vs. Concrete-Observer, Observer vs. Concrete-Subject, Non-participant vs. Concrete-Observer, Non-participant vs. Concrete-Subject and Subject vs. Concrete-Subject. As it can be seen in figure 5.20 that the classes that participate in the Observer role are more cohesive than the classes that participate in the Concrete-Observer and Concrete-Subject roles are more cohesive than the non-participant classes. Also, we can see that the non-participant classes are more cohesive than the classes that participate in the Concrete-Observer and Concrete-Subject roles. Moreover,

we can see that the classes that participate in the Subject role are more cohesive than the classes that participate in the Concrete-Subject role.

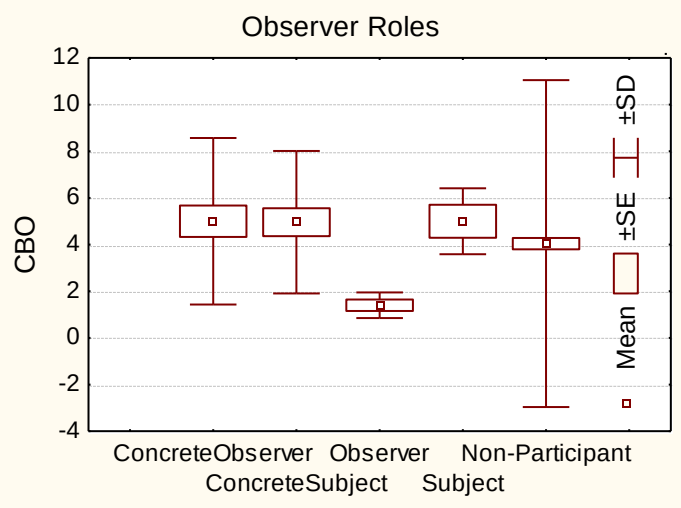
- **Fault-proneness and fault-density**

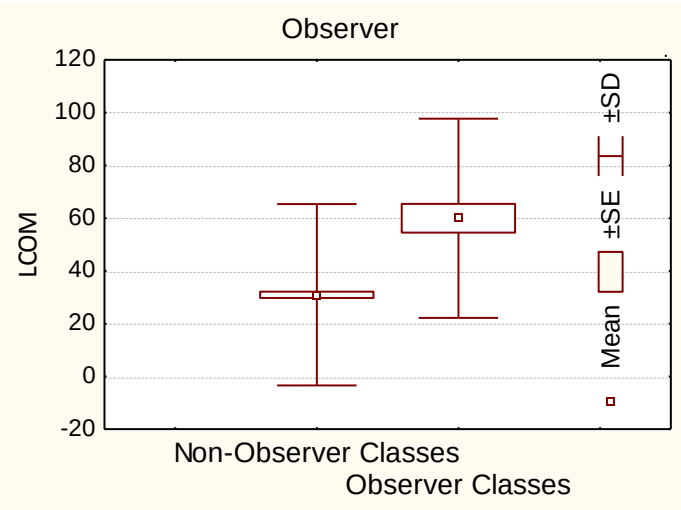
The evaluation of fault-proneness and fault-density of the classes that participate in the Observer design pattern results in insignificant p-values as it can be seen in table 5.22. We can see that the p-values associated with evaluating the difference in fault-proneness and fault-density between the non-Observer and the Observer classes are in significant. Also, we can see that the p-value associated with evaluating the difference in fault-proneness and fault-density among the different roles is not significant as well.

Table 5.22: Evaluation results of Observer pattern and its roles

Observer	
CBO	
Observer classes vs. Non- Observer Classes	**0.005163**
Overall Roles Comparison	**0.001**
Observer vs. Concrete-Observer	**0.016**
Observer vs. Concrete-Subject	**0.012**
Observer vs. Subject	**0.022**
Non-participant vs. Concrete-Observer	**0.012**
Non-participant vs. Concrete-Subject	**0.006**
LCOM	
Observer classes vs. Non- Observer Classes	**0.000000**
Overall Roles Comparison	**0.000**
Observer vs. Non-participant	**0.046**
Observer vs. Concrete-Observer	**0.000**
Observer vs. Concrete-Subject	**0.000**
Non-participant vs. Concrete-Observer	**0.000**
Non-participant vs. Concrete-Subject	**0.000**
Subject vs. Concrete-Subject	**0.036**
Fault-density	

Observer classes vs. Non- Observer Classes	0.149059
Overall Roles Comparison	0.235
Fault-proneness	
Observer classes vs. Non- Observer Classes	0.625757
Overall Roles Comparison	0.467





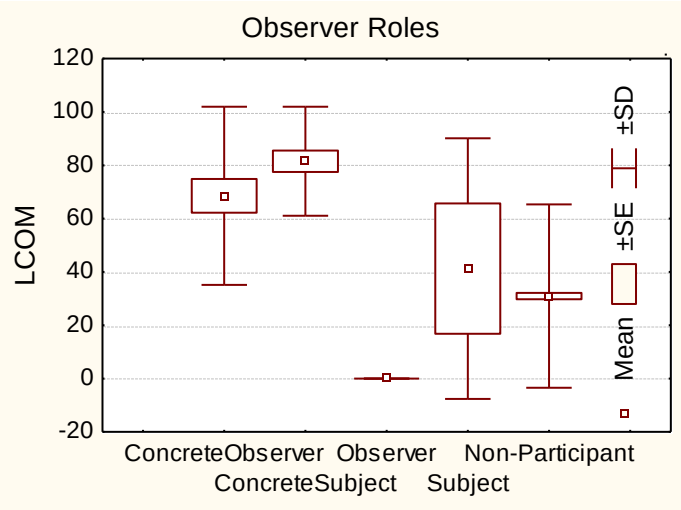


Figure 5.20: Comparison of coupling and cohesion of the Observer design pattern and its roles

5.1.3.3.5 State

- **Coupling**

The p-value associated with evaluating the difference in the coupling of the State-classes and the non-State classes is significant as it can be seen in table 5.23. The result indicates that there is a significant difference in the coupling of the classes that participate in the

State design pattern and the non-participant classes. As it can be seen in figure 5.21, the State-classes are more coupled than the non-State classes.

As it can be seen in table 5.23, the p-value associated with evaluating the difference in the coupling among the classes that participate in the different roles of the State pattern is significant. We can see that there are two pairs of roles that are associated with significant differences. These pairs are: State vs. Context and Non-participant vs. Concrete-State. As it can be seen in figure 5.21 that the classes that participate in the Context role are more coupled than the classes that participate in the State role. Also, we can see that the classes that participate in the Concrete-State are more coupled than the non-participant classes.

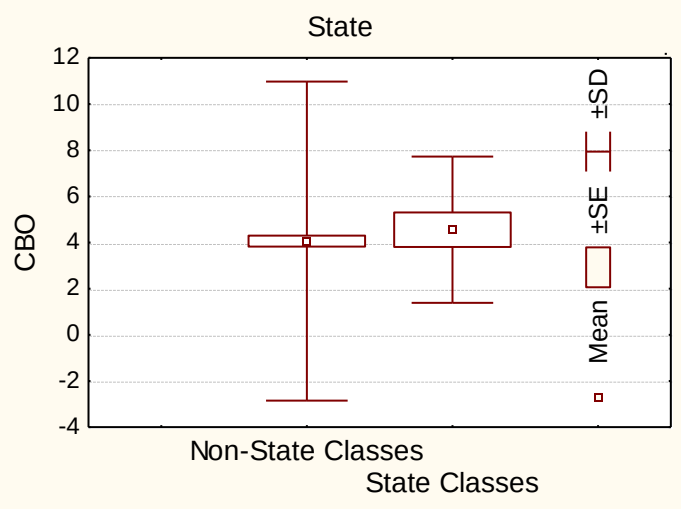
- **Cohesion, fault-proneness and fault-density**

The obtained results show that there are no significant differences. It can be seen in table 5.23 that the p-values associated with the evaluation of cohesion, fault-proneness and fault-density on the pattern level and on the role level are not significant.

Table 5.23: Evaluation results of State pattern and its roles

State	
CBO	
tate classes vs. Non- State Classes	**0.035205**
Overall Roles Comparison	**0.014**
State vs. Context	**0.021**
Non-participant vs. Concrete-State	**0.029**
LCOM	
State classes vs. Non- State Classes	0.205299
Overall Roles Comparison	0.161
Fault-density	
State classes vs. Non- State Classes	0.601959

Overall Roles Comparison	0.238
Fault-proneness	
State classes vs. Non- State Classes	0.542997
Overall Roles Comparison	0.263



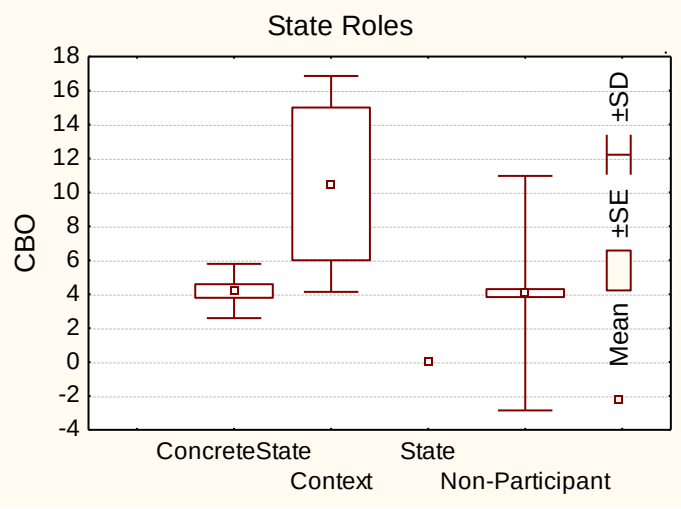


Figure 5.21: Comparison of coupling of the State design pattern and its roles

5.1.3.3.6 Strategy

- **Coupling**

The p-value associated with evaluating the difference in coupling between the Strategy classes and the non-Strategy classes is significant as it can be seen in table 5.24. This

value indicates that there is significant difference in coupling between the Strategy classes and the non-Strategy classes. As it can be seen in figure 5.22, the Strategy classes are more coupled than the non-Strategy classes.

There are significant differences among the classes that participate in the different roles of the Strategy design pattern according to the associated p-value in table 5.24. It can be seen in table 5.24 that there are three pairs of roles that show significant differences. These pairs of roles are: Strategy vs. Context, Non-participant vs. Context and Concrete-Strategy vs. Context. As it can be seen in figure 5.22, the classes that participate in the Context classes are more coupled than the classes that participate in the other roles in the Strategy design pattern and more coupled than the non-participant classes.

- **Cohesion**

The p-value associated with evaluating the difference in cohesion between the classes that participate in the Strategy design pattern and the non-participant classes is significant as shown in table 5.24. This value indicates that there is significant difference between the Strategy and the non-Strategy classes. As it can be seen in figure 5.22, the classes that participate in the Strategy classes are less cohesive than the non-participant classes.

In evaluating the cohesion of the classes that participate in the different roles of the Strategy design pattern, it was found that there is significant difference among the classes that participate in the different roles as it can be seen in table 5.24. We can see that there are five pairs of roles that show significant difference. These pairs are: Strategy vs. Non-participant, Strategy vs. Concrete-Strategy, Strategy vs. Context, Non-participant vs. Concrete-Strategy and Non-participant vs. Context. As it can be seen in figure 5.22, the

classes that participate in the Strategy role are more cohesive than the classes that participate in the other roles of the Strategy design pattern and more cohesive than the non-participant classes. Also, we can see that the non-participant classes are more cohesive than the classes that participate in the Context and Concrete-Strategy roles.

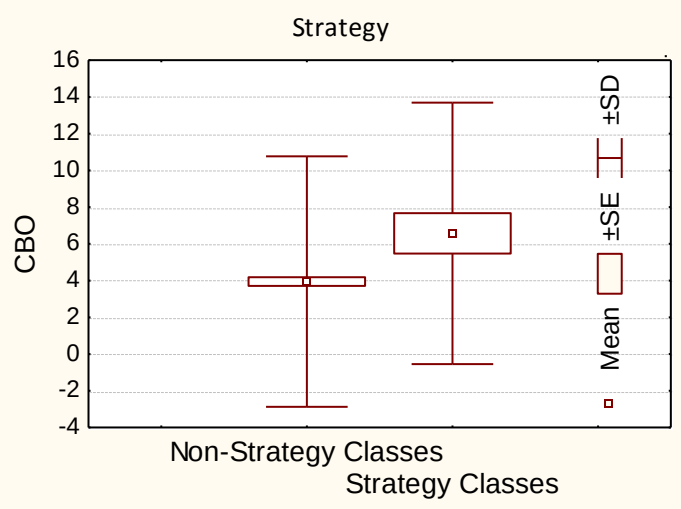
- **Fault-proneness and fault-density**

It can be seen in table 5.24 that the p-values associated with evaluating fault-proneness and fault-density are insignificant on both levels - the pattern level and the role level. There are no significant differences in fault-proneness and fault-density of the classes that participate in the Strategy design pattern compared to the non-participant classes. Also, we can see that there are no significant differences among the different roles of the Strategy design pattern.

Table 5.24: Evaluation results of Strategy pattern and its roles

Strategy	
CBO	
Strategy classes vs. Non-Strategy Classes	**0.003472**
Overall Roles Comparison	**0.000**
Strategy vs. Context	**0.000**
Non-participant vs. Context	**0.000**
Concrete-Strategy vs. Context	**0.004**
LCOM	
Strategy classes vs. Non-Strategy Classes	**0.000000**
Overall Roles Comparison	**0.000**
Strategy vs. Non-participant	**0.026**
Strategy vs. Concrete-Strategy	**0.000**
Strategy vs. Context	**0.000**
Non-participant vs. Concrete-Strategy	**0.000**
Non-participant - Context	**0.000**
Fault-density	

Strategy classes vs. Non-Strategy Classes	0.094428
Overall Roles Comparison	0.182
Fault-proneness	
Strategy classes vs. Non-Strategy Classes	0.513023
Overall Roles Comparison	0.309



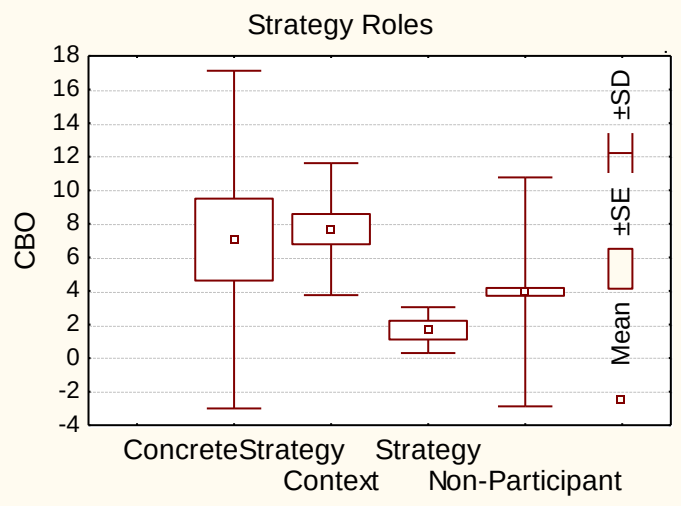


Figure 5.22: Comparison of coupling and cohesion of the Strategy design pattern and its roles

5.1.3.3.7 Template Method

- **Coupling**

The result of evaluating the difference in the coupling between the Template Method classes and the non- Template Method classes indicates that there is a significant difference. The p-value associated with evaluating the difference in coupling between the Template Method classes and the non- Template Method class is significant as in table

5.25. As it can be seen in figure 5.23 that the Template Method classes are more coupled than the non-Template Method classes.

In evaluating the difference in the coupling among the classes that participate in the different roles of the Template Method design pattern, it was found that there is significant difference among the different roles. The p-value associated with evaluating the difference in coupling among the different roles is significant as it can be seen in table 5.25. There are two pairs that show significant differences. These pairs are: Non-participant vs. Concrete-Class and Non-participant vs. Abstract-Class. As it can be seen in figure 5.23, the non-participant classes are less coupled than the classes that participate in the Abstract-Class and the Concrete-Class roles.

- **Cohesion**

The p-value associated with evaluating the difference in cohesion between the Template Method classes and the non-Template Method classes is significant as it can be seen in table 5.25. There is significant difference in cohesion between the classes that participate in the Template Method and the non-Template classes. As it can be seen in figure 5.23, the Template Method classes are less cohesive than the non-Template Method classes.

The same thing can be said in evaluating the difference in cohesion among the classes that participate in the different roles of the Template Method design pattern. There is significant difference among the different roles of the Template Method design pattern as the associated p-value indicates. Two pairs of roles show significant differences. These pairs are: Non-participant vs. Concrete-Class and Non-participant vs. Abstract-Class. As

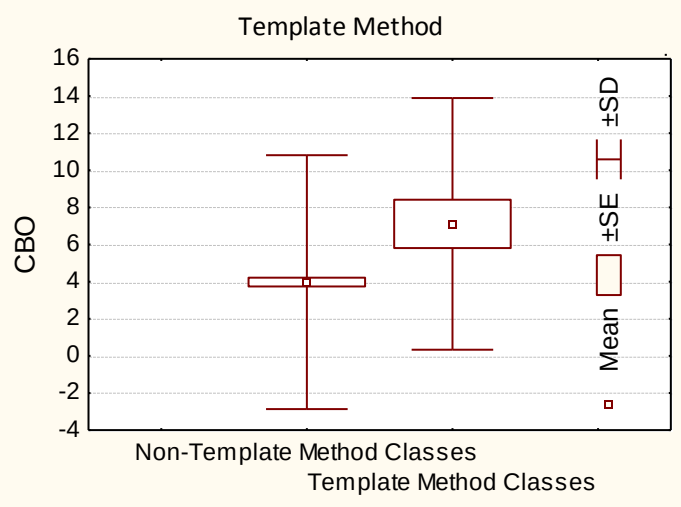
it can be seen in figure 5.23, the non-participant classes are more cohesive than the classes that participate in the Concrete-Class and the Abstract-Class roles.

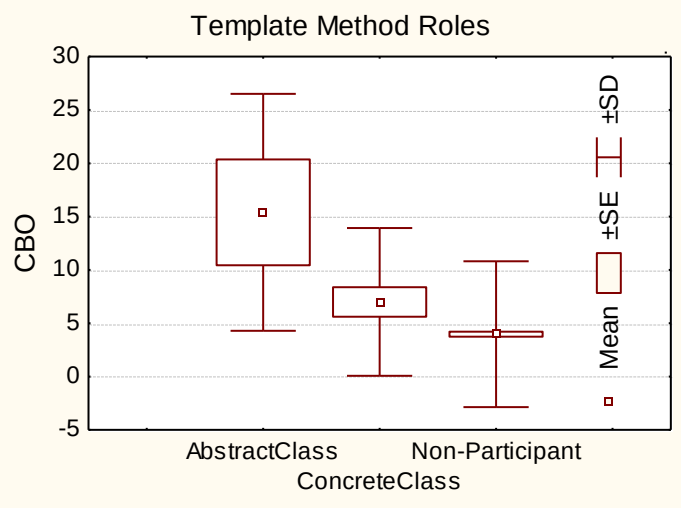
- **Fault-proneness and Fault-density**

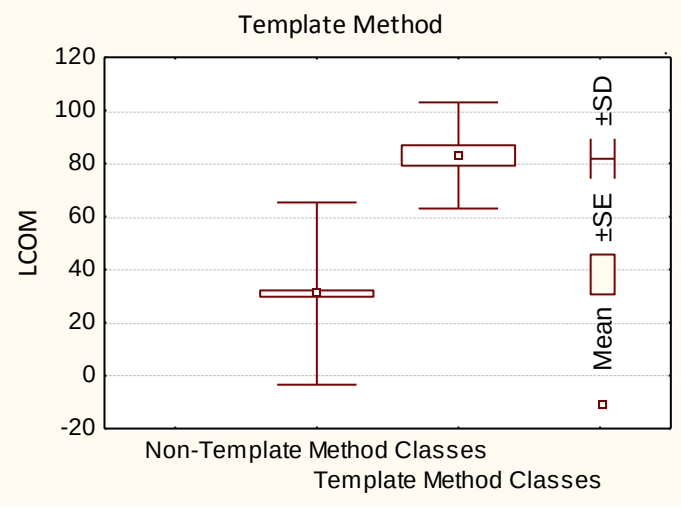
The p-values associated with evaluating the differences in fault-proneness and fault-density are insignificant as in table 5.25. There are no significant differences in fault-proneness and fault-density between the Template Method classes and the non-Template Method classes. Also, we can see that there is no significant difference among the classes that participate in the different roles of the Template Method design pattern.

Table 5.25: Evaluation results of Template Method pattern and its roles

Template Method	
CBO	
Template Method classes vs. Non-Template Classes	**0.000369**
Overall Roles Comparison	**0.000**
Non-participant vs. Concrete-Class	**0.001**
Non-participant vs. Abstract-Class	**0.001**
LCOM	
Template Method classes vs. Non-Template Classes	**0.000000**
Overall Roles Comparison	**0.000**
Non-participant vs. Concrete-Class	**0.000**
Non-participant vs. Abstract-Class	**0.001**
Fault-density	
Template Method classes vs. Non-Template Classes	0.575832
Overall Roles Comparison	0.725
Fault-proneness	
Template Method classes vs. Non-Template Classes	0.581215
Overall Roles Comparison	0.218


Template Method
CBO
16
14
12
10
8
6
4
2
0
-2
-4
±SD
±SE
Mean
Non-Template Method Classes
Template Method Classes


Template Method Roles
CBO
30
25
20
15
10
5
0
-5
±SD
±SE
Mean
AbstractClass
Non-Participant
ConcreteClass


Template Method
LCOM
120
100
80
60
40
20
0
-20
±SD
±SE
Mean
Non-Template Method Classes
Template Method Classes

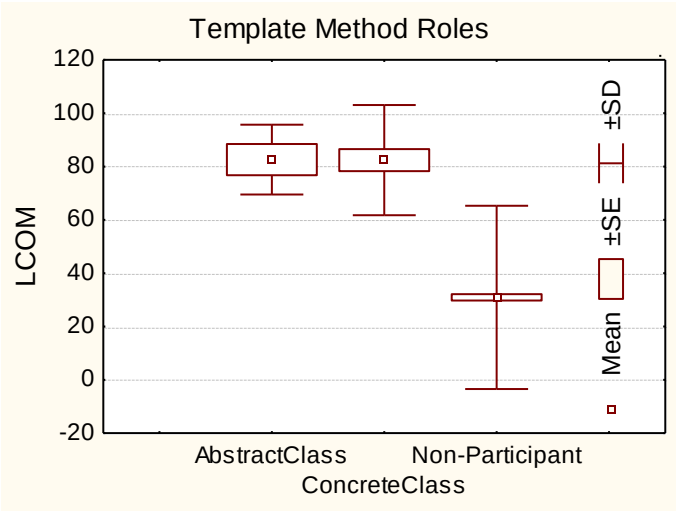


Figure 5.23: Comparison of coupling and cohesion of the Template design pattern and its roles

5.1.3.3.8 Visitor

- **Coupling**

There is a significant difference in the coupling between the Visitor classes and the non-Visitor classes. The p-value associated with evaluating the difference in the coupling between the classes that participate in the Visitor pattern and the non-participant classes is significant as it can be seen in table 5.26. As it can be seen in figure 5.24, the Visitor classes are more coupled than the non-Visitor classes.

In evaluating the difference in the coupling among the different roles of the Visitor design pattern, it was found that there is no significant difference. However, the associated figure 5.24 show that there is a difference but it is not significant.

- **Cohesion, fault-proneness and Fault-density**

The p-values associated with evaluating the difference in cohesion, fault-proneness and fault-density are not significant as it can be seen in table 5.26. There is no significant

difference in cohesion, fault-proneness and fault-density between the Visitor and the non-Visitor classes. Also, we can see that there is no significant difference among the classes that participate in the different roles of the Visitor design pattern as well.

Table 5.26: Evaluation results of Visitor pattern and its roles

Visitor	
CBO	
Visitor classes vs. Non- Visitor Classes	**0.024223**
Overall Roles Comparison	0.075
LCOM	
Visitor classes vs. Non- Visitor Classes	0.318264
Overall Roles Comparison	0.289
Fault-density	
Visitor classes vs. Non- Visitor Classes	0.994465
Overall Roles Comparison	0.407
Fault-proneness	
Visitor classes vs. Non- Visitor Classes	0.892951
Overall Roles Comparison	0.427

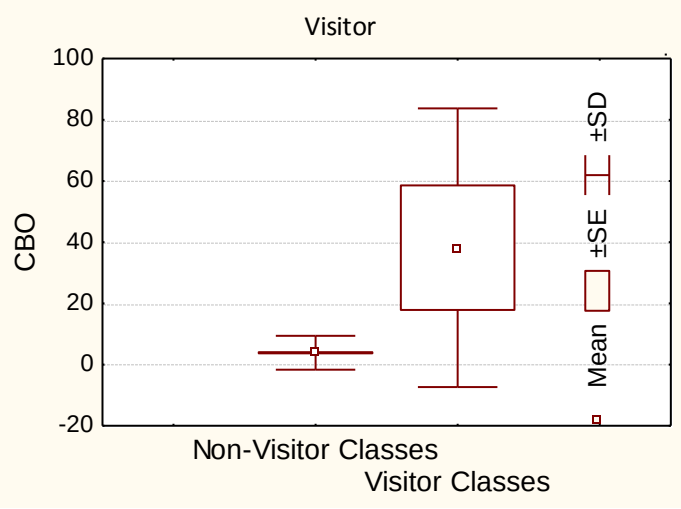


Figure 5.24: Comparison of coupling of the Visitor design pattern and its roles

5.2 Assessing the effectiveness of patterns metrics in fault-prediction

In this section we evaluate the performance of design patterns metrics in faults-prediction. To do so, these metrics are evaluated and compared with CK metrics. We used five design patterns metrics. These metrics are as follows:

- A class is participating in a pattern or not?
- A class participating in a Creational design pattern or not?
- A class participating in a Structural design pattern or not?
- A class participating in a Behavioral design patterns or not?
- In how many patterns a class is participating?

We build two models with these metrics: logistic regression model and linear regression model. Then we build two models (logistic regression model and linear regression model) with CK metrics (CBO, LCOM, RFC, NOC, WMC and DIT). After that we combine both metrics (patterns metrics and CK metrics) and we build two more models (logistic regression model and linear regression model) with this combination.

After building these models, we train and test each model five times. Each time we combine four systems for training and use the fifth for testing.

5.2.1 Fault-proneness prediction

In evaluating the performance of logistic regression models in fault-proneness prediction, two performance measures are used. These measures are CCR (Correct Classification Rate) and AUC (Area Under the Curve). It was found that the patterns model performed worse than the CK metrics model in all cases except in one case (PMD) with respect to CCR. Also, it was found that the performance of CK metrics is degraded when both groups of metrics combined together except in one case (PMD) as it can be seen in table 5.27.

Table 5.27: Performance of CK metrics and Pattern metrics in fault-proneness evaluation.

		Models					
		C&K Metrics		Pattern Metrics		C&K and Pattern Metrics	
Training set	**Test set**	**CCR**	**AUC**	**CCR**	**AUC**	**CCR**	**AUC**
All except JHotDraw	**JHotDraw**	56.77	0.572	50.96	0.507	47.09	0.544
All except JUnit	**JUnit**	75.64	0.638	53.84	0.432	73.07	0.667
All except Lexi	**Lexi**	83.33	0.93	54.16	0.182	79.16	0.86
All except Nutch	**Nutch**	57.57	0.616	55.15	0.493	55.75	0.593
All except PMD	**PMD**	49.32	0.526	47.53	0.502	54.70	0.56

5.2.2 Fault-density prediction

In evaluating the performance of linear regression models in fault-density prediction, three performance measures are used. These measures are MAE (Mean Absolute Error), RMSE (Root Mean Squared Error) and SDAE (Standard Deviation of Absolute Error). CK metrics performed better than patterns metrics in three cases (i.e. JHotDraw, JUnit

and PMD) and performed worse in two cases (Lexi and Nutch). Also, there was degradation in the performance when both groups of metrics combined together compared to the best value when each group evaluated alone except in one case (PMD) as it can be seen in table 5.28.

Table 5.28: Performance of CK metrics and Pattern metrics in fault-density evaluation

		Models								
		CK metrics			**pattern metrics**			**CK & Participation metrics**		
Training set	**Test set**	**MAE**	**RMSE**	**SDAE**	**MAE**	**RMSE**	**SDAE**	**MAE**	**RMSE**	**SDAE**
All except JHotDraw	**JHotDraw**	25.85	37.01	26.57	34.04	47.13	32.70	32.65	43.02	28.10
All except JUnit	**JUnit**	27.55	46.27	37.41	24.53	43.76	36.47	24.61	43.41	35.99
All except Lexi	**Lexi**	14.17	16.44	8.51	26.56	28.90	11.64	21.56	23.82	10.35
All except Nutch	**Nutch**	17.97	21.77	12.31	23.54	26.46	12.11	20.20	23.60	12.23
All except PMD	**PMD**	35.38	66.47	56.33	33.84	66.48	57.28	34.94	65.33	55.26

CHAPTER 6

SUMMARYAND DISCUSSION

In this chapter we summarize and discuss the obtained results in the experimental phase. As we mentioned before, the objectives of our study is to evaluate the modularity and functional correctness of design patterns and to assess the effectiveness of design pattern metrics in fault-prediction.

6.1 Modularity and functional correctness evaluation

6.1.1 Design Level

We evaluate the difference in coupling, cohesion, fault-density and fault-proneness on the design level to measure the impact of design patterns on these attributes in general and to answer RQ1. As a result it was found that the classes that participate in the design patterns are more coupled and less cohesive than the non-participant classes as it can be seen in table 6.1.

The structures of design patterns can justify the increase in coupling and the decrease in cohesion. We know that the classes that participate in the design patterns interact with each other in a certain way to serve some purpose. These interactions increase the value of CBO metric which is used to calculate the coupling in this work. Also, we know that some of the classes that participate in the design patterns are responsible only for delegating the operations to the other classes which, in turns, increase the value of LCOM metric which we used to calculate the Lack of Cohesion. These delegations degrade the

cohesion of a class because these delegations do not interact with the other members of the class. Moreover, the concrete classes play a key role in increasing coupling and decreasing cohesion. This is because the concrete classes need to interact with other classes to implement their operations which, in turns, increase the coupling. These classes are responsible for cohesion decreasing as well. This is because these classes work on the member data of the abstract classes.

Regarding fault-density and fault-proneness, it can be seen in table 6.1 that the classes that participate in the design patterns have no clear tendency for the impact on fault-density and fault-proneness. This might due to many things. First, whether this design pattern patterns are used intentionally or unintentionally? Second, whether the programmers who wrote these programs are experienced in design patterns or not? Third, we need to investigate the nature of these systems.

It is mentioned in the documentation of two systems that they used design patterns. For the other three systems, we do not know for sure whether they used design patterns intentionally or unintentionally. This is the only information we have. The other information is not available. So we cannot provide a detailed justification for the relationship between the participation in design pattern and the presence in faults.

Also, we have to mention that we deduce the results in table 6.1 based on table 6.2. We can see that there is no consensus on the impact of design patterns on coupling, cohesion, fault-density and fault-proneness. To be able to deduce a general summary, we allow, at most, one insignificant case anomaly from the derived conclusion and we ask for, at least, one significant case that support the derived conclusion and the rest of the cases must

support the derived conclusion even they are not significant. This is better and more restricted than the majority voting and less restricted than the consensus

Table 6.1: Summary of the difference in the impact of Participant and Non-participant on coupling, cohesion, fault-density and fault-proneness

First group	Coupling	Cohesion	Fault-density	Fault-proneness	Second group
participant	>	<	-	-	Non-participant

Table 6.2: Detailed summary for the results in the design level

First group	Coupling				Cohesion				Fault-density				Fault-proneness				Second group
Direction	>		<		>		<		>		<		>		<		
Significance	S	N	S	N	S	N	S	N	S	N	S	N	S	N	S	N	
Participants (6)	5	0	0	1	0	1	2	3	1	3	1	1	1	1	1	3	Non-participants

S: # of significant cases and N: # of insignificant cases; (#) is the number of applicable cases;

6.1.2 Category Level

We evaluate the difference in coupling, cohesion, fault-density and fault-proneness of the different categories of design patterns to measure the impact of the different categories of designs on modularity and functional correctness and consequently answer RQ2. It was found that the classes that participate in the different categories of design patterns are more coupled and less cohesive than the non-participant classes as it can be seen in table 6.3. This finding supports the conclusion made in section 6.1. Furthermore, it was found that the distribution of coupling and cohesion is the same among the different categories of design patterns except for evaluating Creational vs. Structural. This is because that all the Categories of design patterns are associated with high values of coupling and low values of cohesion. To make easier to understand, assume that the differences in coupling between the non-participant classes and the classes that participate in the Creational,

Structural and Behavioral design patterns are 10, 11 and 12 respectively. We can say that the difference in coupling among the Structural, Creational and Behavioral are 1 or 2.

Regarding fault-density and fault-proneness, it was found that the Structural design patterns are less fault-dense and prone than the classes that participate in the Creational and Behavioral design patterns and less fault-dense and fault-prone than the non-participant classes. This means that the adoption of the structural design patterns results in a more reliable software because these patterns reduce fault-density and fault-proneness. This is might due to the fact that the idea of these patterns are easier to grasp and consequently easier to apply in software design.

The conclusions made in table 6.3 are derived from table 6.4 in the same way as we did with table 6.1 and table 6.2.

Table 6.3: Summary of the difference in the impact of categories of design patterns on coupling, cohesion, fault-density and fault-proneness

First group	Coupling	Cohesion	Fault-density	Fault-proneness	Second group
Creational	>	<	-	-	Non-participant
Structural	>	<	<	<	Non-participant
Behavioral	>	<	-	-	Non-participant
Creational	>	-	>	>	Structural
Creational	-	-	-	-	Behavioral
Structural	-	-	<	<	Behavioral

Table 6.4: Detailed summary for the results on the category level

First group	Coupling		Cohesion		Fault-density		Fault-proneness		Second group
Direction	>	<	>	<	>	<	>	<	

Significance	S	N	S	N	S	N	S	N	S	N	S	N	S	N	S	N	
Creational (4)	4	0	0	0	0	1	3	0	1	1	1	1	1	1	0	2	Non-participants
Structural (5)	1	3	0	1	0	1	3	1	0	0	1	4	0	0	1	4	Non-participants
Behavioral (6)	5	1	0	0	0	0	5	1	1	1	1	3	1	2	1	2	Non-participants
Creational (3)	1	2	0	0	1	0	0	2	1	2	0	0	0	1	1	1	Structural
*Creational(4)	0	1	0	3	1	1	0	1	1	1	1	1	0	2	1	0	Behavioral
**Structural(5)	1	1	2	1	1	1	0	3	0	1	2	1	0	1	1	3	Behavioral

S: # of significant cases and N: # of insignificant cases; (#) is the number of applicable cases; ***** means that the absent cases in evaluating the cohesion and fault-density have no differences between the creational and behavioral groups; ***** means that the absent case in evaluating the fault-density has no differences between the structural and behavioral groups;

6.1.3 Pattern level

To answer RQ3 we need to evaluate the difference in coupling, cohesion, fault-proneness and fault-density between the classes that participate in each pattern and the classes that do not participate in that pattern. It was found that in all the patterns that show significant differences in coupling and cohesion, the classes that participate in each design pattern are more coupled and less cohesive than the classes that do not participate in that pattern as it can be seen in table 6.5. This explains the obtained results in the previous sections.

Regarding fault-proneness and fault-density, it was found that, in all of the structural design patterns that show significant difference (i.e. Adapter, Composite and Decorator) the classes that participate in the design patterns are less fault-dense and prone than the non-participant classes as it can be seen in table 6.5. This explains the obtained results in evaluating the different categories in the previous section. Also, it was found that the behavioral design patterns have no impact on fault-density and fault-proneness. Furthermore, we can see that only 5 patterns out of the 17 addressed patterns have significant impact on fault proneness and fault-density. We can see that the Builder design

pattern has a negative impact on fault-proneness and fault-density but the Factory Method, Adapter, Composite and Decorator patterns have a positive impact on fault-proneness and fault-density. Moreover, we can see that there is no difference in evaluating the fault-density and fault-proneness for the other patterns.

In comparing our results to the results obtained by other studies, it was found some similarities and some differences. First of all, it can be seen in table 3.4 that Ampatzoglou et al. [9] report that the design patterns, in general, have neutral impact on faults. This is similar to the general conclusion we made as it can be seen in table 6.1. However, they reported that there are two exceptions. These exceptions are associated with Adapter and Observer. They found that the Adapter has a negative impact whereas the Observer has a positive impact. This is different from our work; it was found that the Adapter has positive impact whereas the Observer has neutral impact as it can be seen in table 6.5.

The study conducted by Ampatzoglou et al. is the only study that was conducted on Java systems [9]. The other two studies (i.e. [36] and [8] that addressed faults) are conducted on C++ and C# systems respectively.

Vokac studied 5 patterns [36]. They found that the Singleton and the Observer patterns have a negative impact on faults but it was found that these patterns have neutral impact on faults. Also, they found that Factory Method pattern has a positive impact and the Template Method has a neutral impact on faults which are similar to our findings. For the Decorator pattern, the author reported that its impact on faults is not known. In our case, it was found that the Decorator has a positive impact on faults.

Gatrell and Counsell studied the impact of design patterns on faults [8]. They found that the design patterns, in general, have a negative impact on faults but it was found that the design patterns have a neutral impact on faults except for five patterns.

Table 6.5: Summary of the difference in the impact of the different design patterns on coupling, cohesion, fault-density and fault-proneness

First group	Coupling	Cohesion	Fault-density	Fault-proneness	Second group
Creational design patterns					
Builder classes	-	-	>	>	Non-Builder classes
Factory Meth. classes	>	<	<	<	Non- Factory Meth. classes
Prototype classes	>	<	-	-	Non-Prototype classes
Singleton classes	>	-	-	-	Non-Singleton classes
Structural Design Patterns					
Adapter classes	>	-	<	<	Non-Adapter classes
Bridge classes	-	<	-	-	Non-Bridge classes
Composite classes	-	<	<	<	Non-Composite classes
Decorator classes	-	-	<	<	Non-Decorator classes
Proxy classes	-	-	-	-	Non-Proxy classes
Behavioral design patterns					
Command classes	>	-	-	-	Non-Command classes
Iterator classes	-	<	-	-	Non-Iterator classes
Memento classes	>	<	-	-	Non-Memento classes
Observer classes	>	<	-	-	Non-Observer classes
State classes	>	-	-	-	Non-Participant classes
Strategy classes	>	<	-	-	Non-Strategy classes
Template Meth. classes	>	<	-	-	Non-Template Meth. classes
Visitor classes	>	-	-	-	Visitor classes

6.1.4 Role Level

To answer RQ4, we summarize the differences in coupling, cohesion, fault-density and fault-proneness among the different roles of each design pattern.

- **Coupling**

In evaluating the difference in coupling among the different categories, it was found that 9 of the 17 addressed patterns show significant differences among their roles. These differences might be due to two factors. These factors are the design pattern interface classes (i.e. the classes that the client access to use the design patterns) and the Concrete-classes. By investigation the results obtained in evaluating the coupling of the classes that participate in the different roles of the design patterns, it was found that the design pattern interface classes and the Concrete-classes are associated with significant differences in coupling as it can be seen in table 6.6.

We can see that all the design patterns interface classes (i.e. Product, Abstraction, Caretaker, Originator, Subject and Context in State and Strategy) in table 6.6 are more coupled than the corresponding roles except for Target in the Adapter design pattern. This is might be due to the fact that these classes are more accessed by the clients of these patterns which, in turns, increase the value of the CBO metric that we used to calculate coupling. Also, we can see that the Client in the Command pattern access two roles more than the other roles.

Also, we can see that all the Concrete classes and the classes that implement operations (Adaptee and Refined Abstraction) are more coupled than the classes that participate in the corresponding roles except in some cases where the Concrete classes are compared to the design patterns interface classes. The reason might be due to the fact the concrete classes are responsible to implements the operations of the abstract classes so they may need to access other classes to get this operations implemented.

We need to say that the inheritance is not counted in calculating the CBO metric. So, the classes that are connected to a class through inheritance relationship are not affecting the coupling calculation.

Table 6.6: Summary of the difference in the impact of the different roles of design patterns on coupling

First group	Coupling	Second group
Factory Method		
Product	>	Concrete-Creator
Concrete-Creator	>	Non-participant
Concrete-Product	>	Non-participant
Adapter		
Adaptee	>	Non-participant
Client	>	Non-participant
Target	<	Non-participant
Target	<	Adapter
Target	<	Adaptee
Target	<	Client
Bridge		
Abstraction	>	Non-participant
Refined-abstraction	>	Non-participant
Abstraction	>	Implementor
Refined-abstraction	>	Implementor
Command		
Client	>	Non-participant
Client	>	Concrete-command
Client	>	Invoker
Memento		
Caretaker	>	Non-participant
Originator	>	Non-participant
Observer		
Concrete-Observer	>	Observer
Concrete-Subject	>	Observer
Subject	>	Observer
Concrete-Observer	>	Non-participant
Concrete-Subject	>	Non-participant

State		
Context	>	State
Concrete-State	>	Non-participant
Strategy		
Context	>	Strategy
Context	>	Concrete-Strategy
Context	>	Non-participant
Template Method		
Concrete-Class	>	Non-participant
Abstract-Class	>	Non-participant

- **Cohesion**

In evaluating the difference in cohesion among the classes that participate in the different roles of design patterns, it was found that 11 design patterns out of the 17 addressed patterns show significant differences among their roles as shown in table 6.7. There might be due to two factors. The first factor is the Concrete classes and the classes that implement the Abstract classes (i.e. Leaf and Composite in the Composite design pattern). These classes implement the operations of the abstract classes and may work on the attributes of the base classes so they reduce cohesion. This reduction in cohesion is due to the lack of interactions among the members of the concrete classes which is used to calculate the LCOM Metric.

The second factor is the delegation (such as Invoker) and configuration classes (such as Client in the Command and Context in the Strategy). These classes do not have a lot of interactions among their members and consequently reduce the class cohesion.

Table 6.7: Summary of the difference in the impact of the different roles of design patterns on class cohesion

First group	Cohesion	Second group

Factory Method		
Concrete-Creator	<	Creator
Concrete-Creator	<	Product
Concrete-Creator	<	Non-participant
Concrete-Creator	<	Concrete-Product
Prototype		
Client	<	Prototype
Concrete-Prototype	<	Prototype
Concrete-Prototype	<	Non-participant
Bridge		
Concrete-implementer	<	Non-participant
Composite		
Leaf	<	Component
Composite	<	Component
Client	<	Component
Leaf	<	Non-participant
Composite	<	Non-participant
Client	<	Non-participant
Command		
Invoker	<	Concrete-Command
Command	<	Concrete-Command
Client	<	Concrete-Command
Client	<	Receiver
Command	<	Non-participant
Client	<	Non-participant
Iterator		
Concrete-Aggregator	<	Aggregate
Concrete-Aggregator	<	Non-participant
Concrete-Iterator	<	Iterator
Concrete-Iterator	<	Aggregate
Concrete-Iterator	<	Non-participant
Memento		
Originator	<	Non-participant
Observer		
Observer	>	Non-participant
Concrete-Observer	<	Observer

Concrete-Subject	<	Observer
Concrete-Subject	<	Subject
Concrete-Observer	<	Non-participant
Concrete-Subject	<	Non-participant
Strategy		
Strategy	>	Non-participant
Concrete-Strategy	<	Strategy
Strategy	>	Context
Concrete-Strategy	<	Non-participant
Context	<	Non-participant
Template Method		
Concrete-Class	<	Non-participant
Abstract-Class	<	Non-participant

The suggested factors in justifying the obtained results in evaluating the differences in class coupling and cohesion are not enough to provide complete understanding for these differences. This is because these factors are common for both - the design patterns that show significant differences and the design patterns that do not show significant differences. The question that comes to mind in this case: why some patterns show significant difference while others not? We think that the reason for that can be in the context that the patterns are used in. The context, in which the pattern is used, has an impact on the complexity of the operation implementation in the concrete classes and has an impact on the frequency of use. This consequently affects the way we calculate CBO and LCOM

- **Fault-density & Fault-proneness**

It can be seen in table 6.8 and table 6.9 that only few patterns show significant difference in fault-proneness and fault-density among their roles. For fault-density, we can see that

the only one pair of roles shows significant difference. This pair is: Adapter vs. Client. For fault-proneness, we can see that only three pairs of roles show significant differences. These pairs are: Adapter vs. Adaptee, Adapter vs. Client and Concrete-Product vs. Concrete-Creator. All the other differences were associated with evaluating the differences between the non-participant classes and some design pattern roles. This indicates that the differences that are associated with these patterns in the pattern level were due to these roles.

Table 6.8: Summary of the difference in the impact of the different roles of design patterns on Fault-density

First group	Fault-density	Second group
Factory Method		
Concrete-Product	>	Non-participant
Adapter		
Adapter	<	Non-participant
Adapter	<	Client
Composite		
Leaf	<	Non-participant
Decorator		
Concrete-Decorator	<	Non-Participant
Concrete-Component	<	Non-participant

Table 6.9: Summary of the difference in the impact of the different roles of design patterns on fault-proneness

First group	Fault-proneness	Second group
Builder		
Concrete-Builder	>	Non-participant
Factory Method		
Concrete-Product	<	Non-participant
Concrete-Product	<	Concrete-Creator
Adapter		
Adapter	<	Adaptee
Adapter	<	Non-participant

Adapter	<	Client
Composite		
Leaf	<	Non-participant
Decorator		
Concrete-Decorator	<	Non-Participant
Concrete-Component	<	Non-participant

6.2 Assessing the effectiveness of patterns metrics in fault-prediction

The objective of this section is to evaluate how good participation information in predicting faults is and then to answer RQ4.

6.2.1 Fault-proneness prediction

As it can be seen in table 5.27, we calculate the AUC (area under the ROC curve). The AUC helps analyzing the performance of class participation information in predicting faults. We can use the rules in table 6.10 to give an indication on how to interpret the values of AUC [45]. An AUC value that is greater than 0.7 considers practical [46].

We can see that all the obtained values in table 5.27 are less than 0.7. As a result, we conclude that the participation information is not practical in predicting faults. Also, we can see that there is no major difference between the ability of CK and the ability of participation information in predicting faults which means that the class participation is not responsible for the presence of faults.

Table 6.10: AUC values interpretation

AUC	Interpretation
< 0.5	Bad
0.5 <AUC <0.6	Poor
0.6 <AUC <0.7	Fair
0.7 <AUC <0.8	Acceptable

0.8 <AUC <0.9	Excellent
0.9 <AUC < 1	Outstanding

6.2.2 Fault-density prediction

As shown in table 5.28, the obtained results with evaluating fault-density of design patterns metrics are comparable to the obtained results with CK metrics. However, the absence of a well-established threshold for the MAE, or the other measures, that is used in evaluating the performance of these models in fault-density prediction makes it difficult to say whether the pattern metrics are practical or not.

6.3 Threats to validity

6.3.1 Construct validity

The construct validity is concerned with the measures used in the evaluation. In this study, CBO and LCOM are used to evaluate coupling and cohesion respectively. One threat to construct validity can stem from the use of CBO in coupling evaluation. This threat is that the CBO does not take into account the inheritance coupling. This limitation results in lower values for CBO for the classes that participate in an inheritance relationship. However, this does not affect the obtained conclusion. In fact, considering inheritance coupling will lead to further support for the obtained results. This is because the structure of most of design patterns includes inheritance. At least the calculation of inheritance coupling will not lead to significant change in the obtained results this is because both of the classes that participate in the design patterns and the non-participant classes may have inheritance relationships. The LCOM metric suffers from one limitation as well. The LCOM metric does not take into consideration the direct interactions among the data

members and among the operations. However, this is common for both, the participant and the non-participant classes in the design patterns. In addition to the above, the other coupling and cohesion metrics have their own limitations. So, if coupling and cohesion is calculated using different metrics this may result in different problems.

For evaluating functional correctness, we used fault-proneness and density as proxy metrics. The problem with these faults' measures is that they do not differentiate between the severity levels of faults. In our work, all faults are treated the same way in spite of that there could be big differences in their severity.

6.3.2 Internal validity

Internal validity is the degree to which the observed effects depend only on the intended experimental variables. Faults data can be a threat to the internal validity of this study. We collected the already identified faults and there could be other undiscovered faults. But this is not a major issue because these systems are popular, open-source and widely used systems and the level of fault inspection of these systems is high. Another threat to the internal validity is emerge from the developers' background. We do not know for sure whether the developers are trained to work with design patterns or not. However, we are not studying cause-and-effect relationship because we cannot control each variable that affect the relationships among the different groups. We are only trying to see if there is a significant association between the addressed variables or not.

6.3.3 External validity

The external validity is concerned with the generalizability. We identified two threats to the external validity of this study. First, the nature of the subject systems is a threat to validity of this study. All subject systems are open-source systems and developed using

one programming language- Java. To be able to generalize the obtained results in this study we have to investigate the obtained results by including commercial systems and systems developed using other programming languages. However, this study can be considered as a step that can be strengthened later with more replications.

6.3.4 Conclusion validity

Conclusion validity is the degree to which the conclusions that are obtained about the relationships in the tesing data are reasonable. We performed our experiments in the design level and the category level on 6 cases (i.e. 5 subject systems and when all of these systems combined together). The obtained results show that there is no consensus among all of these cases. To be able to draw conclusions on the general tendencies of the addressed attributes (i.e. coupling, cohesion, fault-proneness and fault-density), we propose the criteria used in chapter 6 to draw the obtained conclusions. In these criteria, we allow only for one case anomaly at most, from the direction of the obtained conclusion, which is not associated with significant p-value. Also, there should be at least one case that shows significant difference in the direction of the obtained conclusion. The other cases should be in the same direction of the obtained conclusion. However, this is better and more restricted than the majority voting and less restricted than the consensus. Considering more cases or more systems may lead to different conclusions based on these criteria.

Another threat to the conclusion validity is the use of non-parametric tests (i.e. Mann-Whitney test and Kruskal-Wallis test). The non-parametric tests are less powerful than the parametric tests but they do not assume that the data is normally distributed and we cannot assume normality for the data used in this thesis.

CHAPTER 7

CONCLUSION AND FUTURE WORK

The objective of conducting this study is to evaluate modularity (coupling and cohesion) and functional correctness (fault-proneness and fault-density) of design patterns and to assess the effectiveness of some design patterns metrics in fault-prediction.

To do so, first, the modularity and functional correctness are evaluated and compared for the participant classes versus the non-participant classes on the design level. It was found that the classes that participate in the design patterns are more coupled and less cohesive than the non-participant classes. For fault proneness and density, it was found that there is no clear tendency for the difference between the participant and the non-participant classes. However, we cannot generalize these findings on the lower levels. We need to investigate further and dig deeper to understand the impact of design patterns on coupling, cohesion, fault-density and fault-proneness. This is because the different categories of patterns have different purposes. As we know, the creational design patterns have something to do with object creation and the structural design patterns have something to do with the structure and the behavioral design patterns have something to do with the behavior. So, we needed to investigate the modularity and functional correctness on the category level.

Second, the modularity and functional correctness are evaluated and compared for each category of design patterns (creational, structural and behavioral). It was found that the

classes that participate in the different categories are more coupled and less cohesive than the non-participant classes in these categories. But when the different categories are compared with each other, we found that there are no clear tendencies for the differences among the different categories of design patterns except when we evaluate the coupling of the creational design patterns versus the structural design patterns. For the other pairs, there were no clear tendencies for their differences. This is because all of them are of high coupling and low cohesion. So, there were clear tendency for their differences when they are compared to the non-participant classes, which are associated with lower levels of coupling and higher levels of cohesions, but we did not find clear tendency for their differences when they are compared with each other. For fault-proneness and fault-density, it was found that the classes that participate in the structural design patterns are of less fault proneness and density than the other categories and less than the non-participant classes as well. So, the results suggest that the structural design patterns have a positive impact on functional correctness. All the other pairs show no clear tendencies. However, the obtained results on the category level are not enough to justify the impact of design patterns on modularity and functional correctness. This is because each pattern has its own problem to solve. So, we needed to investigate each pattern.

Third, the modularity and functional correctness are evaluated and compared for each design pattern. In this phase we evaluate modularity and functional correctness for the classes that participate in each design pattern and the non-participant classes in that pattern. It was found that all the classes that participate in the design patterns that show significant difference in evaluating the coupling and cohesion were more coupled and less cohesive than the non-participant classes in those patterns. For the other patterns, we think

that if they were used in a different context they will show significant differences. In evaluating fault density and proneness it was found that only five patterns show significant differences. These patterns are: Builder, Factory Method, Adapter, Composite and Decorator. All of these patterns show that they have a positive impact on fault density and proneness except for Builder which has a negative impact.

Fourth, the modularity and functional correctness are evaluated and compared for the different roles of each design pattern. It was found that the classes that participate in the concrete roles are usually more coupled and less cohesive than the other roles. Also, it was found that the design pattern interface classes are more coupled than the other roles as well. Moreover, it was found that the classes that participate in the delegation and configuration roles are less cohesive than the other roles. For fault-proneness and fault-density, we found that most of the significant differences where associated with comparing the non-participant classes versus the classes that participate in the roles of design patterns. Only three pairs of roles show significant difference in fault-proneness. These pairs are: Adapter vs. Adaptee, Adapter vs. Client and Concrete-Product vs. Concrete-Creator. For fault-density, we found that only one pair of roles shows significant difference. This pair is: Adapter vs. Client.

Next, the effectiveness of design patterns metrics are evaluated and compared with CK metrics with respect to fault-prediction. It was found that the design pattern metrics are not useful in fault-prediction.

7.1 Research Contribution

The contributions of this work can be divided into two parts: the main contributions and the sub-contributions.

7.1.1 Main Contributions

There are three main contributions of this study:

- Empirical evidence on the results of modularity (coupling and cohesion) evaluation of design patterns in object-oriented systems will be provided.
- Empirical evidence on the results of functional correctness (fault-proneness and fault density) evaluation of design patterns in object-oriented systems will be provided.
- Empirical evidence on the effectiveness of some design patterns metrics in faults prediction in object-oriented systems.

7.1.2 Sub-Contributions

A comparative literature survey was conducted to collect the existing empirical evidence to help in understanding the impact of software design patterns on the software quality attributes and to assess the current state of research in this area [47].

7.2 Future work

There are many venues for future works.

First, we performed our experiments on five systems that are developed with java programming language. So, we cannot generalize our results to other systems that are developed in different programming languages. As we know, the different programming

languages have different levels of expressiveness. This might affect the application of design patterns in the different programming languages. Based on that, we cannot generalize the obtained results. To be able to generalize the obtained results on the different programming language we need to replicate these experiments on software systems that are developed in different programming languages.

Second, we performed our experiments on open source software systems. These systems are developed by many developers from different backgrounds. Their levels of patterns experience are not controlled. This is different from the commercial companies where the developers of software systems can be trained and prepared to work with software design patterns. So, we need to replicate these experiments on commercial systems. Doing so will help improve the generalizability of the obtained results.

Third, these experiments are performed on 17 GoF design patterns that participate in five systems. As we know, the number of GoF patterns is 23. The other 6 patterns (i.e. Abstract Factory, Façade, Flyweight, Mediator, Chain-of-responsibility and Interpreter) are not addressed in this study. This is because they are not used in the subject systems we used in this study. So, they can be addressed in the future replications. Not only that, more patterns instances and more subject systems can be considered as well.

Finally, we performed our experiments on object oriented design patterns. In the future, we think of addressing design patterns that are developed in other paradigms such as aspect oriented paradigm [48]. This is because the representations of these patterns are different from one paradigm to the other which might affect their applicability.

References

[1] E. Gamma, *et al.*, *Design Patterns: Elements of Reusable Object-Oriented Software*: Addison-Wesley, 1995.

[2] L. Prechelt, *et al.*, "Two controlled experiments assessing the usefulness of design pattern documentation in program maintenance," *IEEE Transactions on Software Engineering*, vol. 28, pp. 595-606, 2002.

[3] B. Venners. (2005, 16/10/2013). *How to Use Design Patterns - A Conversation With Erich Gamma*. Available: http://www.artima.com/lejava/articles/gammadp.html

[4] W. B. McNatt and J. M. Bieman, "Coupling of design patterns: common practices and their benefits," in *25th Annual International Computer Software and Applications Conference, 2001. COMPSAC 2001.* , 2001, pp. 574-579.

[5] P. Wendorff, "Assessment of design patterns during software reengineering: lessons learned from a large commercial project," in *Fifth European Conference on Software Maintenance and Reengineering, 2001*, 2001, pp. 77-84.

[6] F. Khomh and Y. G. Gueheneuc, "Do Design Patterns Impact Software Quality Positively?," in *12th European Conference on Software Maintenance and Reengineering, 2008. CSMR 2008.* , 2008, pp. 274-278.

[7] Y. G. Guéhéneuc and H. Albin-Amiot, "Using design patterns and constraints to automate the detection and correction of inter-class design defects," in *39th International Conference and Exhibition on Technology of Object-Oriented Languages and Systems, 2001. TOOLS 39.* , 2001, pp. 296-305.

[8] M. Gatrell and S. Counsell, "Design patterns and fault-proneness a study of commercial C# software," in *2011 Fifth International Conference on Research Challenges in Information Science (RCIS)* 2011, pp. 1-8.

[9] A. Ampatzoglou, *et al.*, "An empirical investigation on the impact of design pattern application on computer game defects," presented at the Proceedings of the 15th International Academic MindTrek Conference: Envisioning Future Media Environments, Tampere, Finland, 2011.

[10] Y.-G. Guéhéneuc, "P-mart: Pattern-like micro architecture repository," in *Proceedings of the 1st EuroPLoP Focus Group on Pattern Repositories*, 2007.

[11] ISO, " ISO/IEC CD 25010 Software engineering - Software product Quality Requirements and Evaluation (SQuaRE) Quality model and guide," 2009.

[12] S. R. Chidamber and C. F. Kemerer, "A metrics suite for object oriented design," *IEEE Transactions on Software Engineering*, vol. 20, pp. 476-493, 1994.

[13] Y. G. Guéhéneuc and G. Antoniol, "DeMIMA: A Multilayered Approach for Design Pattern Identification," *IEEE Transactions on Software Engineering*, vol. 34, pp. 667-684, 2008.

[14] D. Jing, *et al.*, "DP-Miner: Design Pattern Discovery Using Matrix," in *14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems, 2007. ECBS '07.* , 2007, pp. 371-380.

[15] A. D. Lucia, *et al.*, "Design pattern recovery through visual language parsing and source code analysis," *Journal of Systems and Software*, vol. 82, pp. 1177-1193, 2009.

[16] J. Niere, *et al.*, "Towards pattern-based design recovery," in *Proceedings of the 24rd International Conference on Software Engineering, 2002. ICSE 2002.* , 2002, pp. 338-348.

[17] F. Arcelli Fontana and M. Zanoni, "A tool for design pattern detection and software architecture reconstruction," *Information Sciences*, vol. 181, pp. 1306-1324, 2011.

[18] S. Nija and R. A. Olsson, "Reverse Engineering of Design Patterns from Java Source Code," in *21st IEEE/ACM International Conference on Automated Software Engineering, 2006. ASE '06.* , 2006, pp. 123-134.

[19] Y.-G. Guéhéneuc, "Ptidej: Promoting Patterns with Patterns," in *Proceedings of the first ECOOP workshop on Building a System using Patterns*, Glasgow, UK,, 2005.

[20] N. Tsantalis, *et al.*, "Design Pattern Detection Using Similarity Scoring," *IEEE Transactions on Software Engineering*, vol. 32, pp. 896-909, 2006.

[21] J. M. Smith and D. Stotts, "SPQR: flexible automated design pattern extraction from source code," 2003, pp. 215-224.

[22] J. Dietrich and C. Elgar, "Towards a web of patterns," *Web Semantics: Science, Services and Agents on the World Wide Web*, vol. 5, pp. 108-116, 2007.

[23] L. Prechelt, *et al.*, "A controlled experiment in maintenance: comparing design patterns to simpler solutions," *IEEE Transactions on Software Engineering*, vol. 27, pp. 1134-1144, 2001.

[24] M. Vokac, *et al.*, "A Controlled Experiment Comparing the Maintainability of Programs Designed with and without Design Patterns—A Replication in a Real Programming Environment," *Empirical Software Engineering*, vol. 9, pp. 149-195, 2004.

[25] L. Prechelt and M. Liesenberg, "Design Patterns in Software Maintenance: An Experiment Replication at Freie Universität Berlin," in *2nd International Workshop on Replication in Empirical Software Engineering Research (RESER), 2011* 2011, pp. 1-6.

[26] N. Juristo and S. Vegas, "Design Patterns in Software Maintenance: An Experiment Replication at UPM - Experiences with the RESER'11 Joint Replication Project," in *2nd International Workshop on Replication in Empirical Software Engineering Research (RESER), 2011* 2011, pp. 7-14.

[27] A. Nanthaamornphong and J. C. Carver, "Design Patterns in Software Maintenance: An Experiment Replication at University of Alabama," in *2nd International Workshop on Replication in Empirical Software Engineering Research (RESER), 2011* 2011, pp. 15-24.

[28] J. L. Krein, *et al.*, "Design Patterns in Software Maintenance: An Experiment Replication at Brigham Young University," in *2nd International Workshop on Replication in Empirical Software Engineering Research (RESER), 2011* 2011, pp. 25-34.

[29] J. Garzás and M. Piattini, "Do rules and patterns affect design maintainability?," *Journal of Computer Science and Technology*, vol. 24, pp. 262-272, 2009.

[30] S. Jeanmart, *et al.*, "Impact of the visitor pattern on program comprehension and maintenance," in *3rd International Symposium on Empirical Software Engineering and Measurement, 2009. ESEM 2009.* , 2009, pp. 69-78.

[31] L. Aversano, *et al.*, "An empirical study on the evolution of design patterns," presented at the Proceedings of the the 6th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering, Dubrovnik, Croatia, 2007.

[32] M. Gatrell, *et al.*, "Design Patterns and Change Proneness: A Replication Using Proprietary C# Software," in *16th Working Conference on Reverse Engineering, 2009. WCRE '09.* , 2009, pp. 160-164.

[33] J. M. Bieman, *et al.*, "Design patterns and change proneness: an examination of five evolving systems," in *Proceedings of the 9th International Software Metrics Symposium, 2003.* , 2003, pp. 40-49.
[34] T. Afacan, "State Design Pattern Implementation of a DSP processor: A case study of TMS5416C," in *6th IEEE International Symposium on Industrial Embedded Systems (SIES), 2011*, 2011, pp. 67-70.
[35] J. Rudzki, "How design patterns affect application performance – A Case of a Multi-tier J2EE Application," presented at the Proceedings of the 4th international conference on Scientific Engineering of Distributed Java Applications, Luxembourg-Kirchberg, Luxembourg, 2005.
[36] M. Vokac, "Defect frequency and design patterns: an empirical study of industrial code," *IEEE Transactions on Software Engineering*, vol. 30, pp. 904-917, 2004.
[37] F. Arcelli Fontana, *et al.*, "Understanding the relevance of micro-structures for design patterns detection," *Journal of Systems and Software*, vol. 84, pp. 2334-2347, 2011.
[38] A. De Lucia, *et al.*, "Behavioral Pattern Identification through Visual Language Parsing and Code Instrumentation," in *13th European Conference on Software Maintenance and Reengineering, 2009. CSMR '09.* , 2009, pp. 99-108.
[39] A. De Lucia, *et al.*, "Improving Behavioral Design Pattern Detection through Model Checking," in *2010 14th European Conference on Software Maintenance and Reengineering (CSMR)*, 2010, pp. 176-185.
[40] M. L. Bernardi, *et al.*, "A model-driven graph-matching approach for design pattern detection," in *20th Working Conference on Reverse Engineering (WCRE), 2013* 2013, pp. 172-181.
[41] Y.-G. Guéhéneuc, *et al.*, "Improving design-pattern identification: a new approach and an exploratory study," *Software Quality Control*, vol. 18, pp. 145-174, 2010.
[42] SciTools. (2014, 15/1/2014). *http://www.scitools.com/download/*.
[43] H. B. Mann and D. R. Whitney, *On a Test of Whether One of Two Random Variables is Stochastically Larger Than the Other*: Institute of Mathematical Statistics, 1947.
[44] W. H. Kruskal and W. A. Wallis, "Use of Ranks in One-Criterion Variance Analysis " *Journal of the American Statistical Association*, vol. Vol. 47, pp. 583-621, 1952.
[45] H. Jr, *et al.* (2013). *Applied logistic regression*.
[46] R. Shatnawi, *et al.*, "Finding software metrics threshold values using ROC curves," *Journal of Software Maintenance and Evolution: Research and Practice* vol. 22, pp. 1-16, 2010.
[47] M. Ali and M. O. Elish, "A Comparative Literature Survey of Design Patterns Impact on Software Quality," in *Information Science and Applications (ICISA), 2013 International Conference on*, 2013, pp. 1-7.
[48] G. Kiczales, *et al.*, "Aspect-oriented programming," in *ECOOP'97 — Object-Oriented Programming*. vol. 1241, M. Akşit and S. Matsuoka, Eds., ed: Springer Berlin Heidelberg, 1997, pp. 220-242.

Vitae

Name	: Mawal Ali Abdo Mohammed
Nationality	: Yemeni
Date of Birth	:6/15/1984
Email	: mawal.mohammed@yahoo.com
Address	: Taiz - Yemen
Academic Background	: Received Bachelor degree in Software engineering from Taiz University
Work experience	:University of Taiz, Faculty of Engineering, Teaching assistant (Lab instructor), Dec, 2008 – August, 2011
Published works	:Mawal Ali, Mahmoud O. Elish, "A Comparative Literature Survey of Design Patterns Impact on Software Quality," 2013 International Conference on Information Science and Applications (ICISA), pp. 1-7, 2013